Bayern | NTG | Grundlegendes Niveau

INFORMATIK 6

Listen | Bäume | Rekursion
Nebenläufigkeit und Threads
Informationssicherheit
Softwareprojekte

Material online verfügbar

Cornelsen

INFORMATIK 6

NTG | Grundlegendes Niveau

Listen | Bäume | Rekursion
Nebenläufigkeit und Threads
Informationssicherheit
Softwareprojekte

Autoren
Peter Brichzin (München)
Florian Janus (Oettingen i. Bay.)
Franz Jetzinger (München)
Johannes Neumeyer (München)
Klaus Reinold (München)
Dr. Stefan Seegerer (Nürnberg)
Albert Wiedemann (Oberottmarshausen)

Berater
Dennis Stanglmair (Mainburg)

Die enthaltenen Links verweisen auf digitale Inhalte, die der Verlag bei verlagsseitigen Angeboten in eigener Verantwortung zur Verfügung stellt. Links auf Angebote Dritter wurden nach den gleichen Qualitätskriterien wie die verlagsseitigen Angebote ausgewählt und bei Erstellung des Lernmittels sorgfältig geprüft. Für spätere Änderungen der verknüpften Inhalte kann keine Verantwortung übernommen werden.

Cornelsen

Vorwort 4

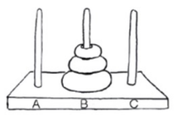
Startklar 7

1 Die rekursive Datenstruktur Liste 13

1.1 Prototyp 1: Die rekursive Warteschlange 14
1.2 Prototyp 2: Trennung von Struktur und Daten 18
*1.3 Prototyp 2+: Bessere Trennung hilft beim Recycling 24
1.4 Prototyp 3: Rekursive Methoden der Liste 26
Einschub: Programme testen 30
Einschub: Prototyping 31
1.5 Endprodukt: Liste mit Entwurfsmuster Kompositum 32
1.6 Stapel und Warteschlange: Spezialformen der Liste 42
Teste dich selbst 46
Zusammenfassung 49
Zum Weiterlesen
L1 Termauswertung 50

2 Die rekursive Datenstruktur Baum 51

2.1 Effiziente Suchen unterstützen: Datenstruktur Baum 52
2.2 Suchen und Einfügen: Rekursion im geordneten Binärbaum 58
2.3 Entwurfsmuster nutzen: Binärbaum implementieren 62
2.4 Binärbäume durchlaufen: Information aller Daten abfragen 70
2.5 Üben und Vertiefen: Weiterführende Aufgaben 74
Teste dich selbst 80
Zusammenfassung 82
Zum Weiterlesen
L2 Übersicht Datenstrukturen 84
L3 AVL -Bäume 85



3 Rekursion 87

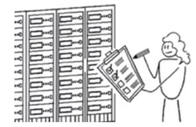
3.1 Methoden rufen sich selbst auf: Rekursion 88
3.2 Anwendung der Rekursion: Backtracking 96
3.3 Alle Knoten eines Graphen besuchen: Tiefensuche 102
3.4 Anwendungen der Tiefensuche: Optimale Lösungen 108
Teste dich selbst 112
Zusammenfassung 113
Zum Weiterlesen
L4 Fraktale – Selbstähnlichkeit 115
L5 Rekursion und Kellerspeicher 116

4 Nebenläufigkeit und Threads 117

4.1 Nebenläufigkeit: Prozesse und Threads 118
4.2 Wechselseitiger Ausschluss 122
4.3 Eine gemeinsame Ressource: Effiziente Synchronisierung 128
4.4 Mehrere gemeinsame Ressourcen: Deadlocks verhindern 134
Teste dich selbst 140
Zusammenfassung 142
Zum Weiterlesen
L6 Verschiedene Formen der Parallelität 144
L7 Kooperative Threads 145

5 Informationssicherheit 147

5.1 Schwachstellen analysieren: Gefahren für IT-Systeme 148
5.2 Systeme schützen: Maßnahmen der IT-Sicherheit 152
Zum Weiterlesen
L8 Hacks und Datenlecks 156
L9 Das Bundesamt für Sicherheit in der Informationstechnik 157
L10 Glossar wichtiger Begriffe 158



6 Softwareentwicklung 159

6.1 Im Team arbeiten: Agile Methoden 160
6.2 Projekte organisieren: Das Wasserfallmodell 164
6.3 Das Rad nicht neu erfinden: Bibliotheken nutzen 166
6.4 Grafische Benutzeroberflächen 168
6.5 Daten persistent speichern: Datenbank oder Datei 170
6.6 Das Entwurfsmuster MVC 175
6.7 Debuggen und Testen 178
6.8 Softwarequalität aus Entwicklersicht 182
6.9 Versionsverwaltungssysteme 183
6.10 Projektvorschläge 185

Anhang 187

Lösungen zu „Teste dich selbst“ 187
Stichwortverzeichnis 195
Bildquellenverzeichnis 198
Textquellenverzeichnis 199

Vorwort

Liebe Informatikerin, lieber Informatiker der nächsten Generation, damit Sie sich bei der Arbeit mit dem Buch gut zurechtfinden, wird hier der Aufbau kurz beschrieben.

4 Nebenläufigkeit und Threads

4.2 Wechselseitiger Ausschluss

Die SMV möchte einen Schulball veranstalten. Hierfür sollen im Vorfeld Getränke, Snacks und Dekorationsartikel eingekauft werden. Für diese Einkäufe steht ein Budget von 500 Euro bereit. Der Kassenwart der SMV schlägt für einen möglichst effizienten Ablauf folgendes Vorgehen vor:
- Der Einkauf wird auf mehrere SMV-Mitglieder aufgeteilt (ein Mitglied kauft Getränke, ein anderes Mitglied die Snacks usw.).
- Damit das Budget nicht überschritten wird, müssen sich alle einkaufenden Mitglieder zunächst beim Kassenwart erkundigen, wie viel Geld noch übrig ist, bevor sie zu ihrem Einkauf aufbrechen.
- Nachdem die Einkäufer zurückkehren, reichen sie ihre Quittung beim Kassenwart ein und erhalten ihre Auslagen erstattet. Das Restbudget verringert sich dadurch entsprechend.

Gesagt getan! Doch es stellt sich heraus, dass die Einkäufer das Budget von 500 Euro bei weitem überschritten haben, obwohl sich alle an das vereinbarte Vorgehen gehalten haben.

a Erläutern Sie, wie es zu der Budgetüberschreitung kommen konnte.
b Stellen Sie einen möglichen Verlauf, der zur Budgetüberschreitung führt, als Sequenzdiagramm dar. Dieses soll neben dem Kassenwart zwei Einkäufer enthalten.
c Diskutieren Sie verschiedene Möglichkeiten, wie das Verfahren angepasst werden könnte, um eine Budgetüberschreitung vollständig auszuschließen.

Signale sorgen bei der Eisenbahn für Sicherheit

Im Alltag kennt man nebenläufige Abläufe z. B. von der Eisenbahn. Im Schienennetz bewegen sich zur gleichen Zeit viele verschiedene Züge weitgehend unabhängig voneinander. Um Unfälle zu vermeiden, muss jedoch sichergestellt werden, dass ein Streckenabschnitt nur von einem Zug gleichzeitig befahren wird. Dafür kommen Signale entlang der Bahnstrecke zum Einsatz, welche die Einfahrt in einen Streckenabschnitt nur dann gestatten, wenn dort noch kein Zug fährt. Nach der Einfahrt eines Zuges schalten die Signale deshalb auf Rot und zwingen so alle eventuell folgenden Züge zum Warten, bis der erste Zug den Streckenabschnitt wieder verlassen hat. Erst dann wird das Signal auf Grün zurückgesetzt, sodass der nächste Zug in den Abschnitt einfahren kann.

Synchronisation von Threads und Prozessen

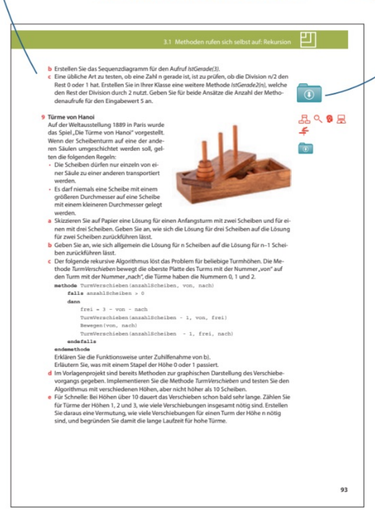
Auch bei nebenläufigen Programmen kommt es häufig vor, dass etwa bestimmte Ressourcen nur von einem Thread gleichzeitig genutzt werden können. Ein Beispiel hierfür ist die in Smartphones verbaute Hauptkamera. Würden mehrere Threads gleichzeitig auf die Kamera zugreifen, könnte es schnell passieren, dass ein Thread der Kamera den Befehl zum Herauszoomen gibt, während zeitgleich ein anderer Thread den Befehl zum Herauszoomen gibt. Aber auch beim unkontrollierten nebenläufigen Zugriff auf gemeinsam genutzte Speicherinhalte kann es zu Problemen kommen, wie das folgende Beispiel einer vereinfachten Bankkontoverwaltung zeigt. Die Klasse BANKKONTO fungiert dabei als Schnittstelle zur Datenbank mit den Kontoinformationen,

122

Inhaltliche Kapitel mit handlungsorientierten Einstiegsaufgaben, Erklärungen und Beispielen, damit Sie informatische Konzepte verstehen und kreativ einsetzen können

Vielfältige Aufgaben zum Üben und Vertiefen sowie spannende Forschungsaufträge als Blick über den Tellerrand und Lehrplan hinaus

Dateivorlagen auch im Internet unter informatikschulbuch.de


3.1 Methoden rufen sich selbst auf: Rekursion

b Erstellen Sie das Sequenzdiagramm für den Aufruf istGerade(3).
c Eine übliche Art zu testen, ob eine Zahl n gerade ist, ist zu prüfen, ob die Division n/2 den Rest 0 oder 1 hat. Erstellen Sie in Ihrer Klasse eine weitere Methode istGerade2(n), welche den Rest der Division durch 2 nutzt. Geben Sie für beide Ansätze die Anzahl der Methodenaufrufe für den Eingabewert 5 an.

9 Türme von Hanoi

Auf der Weltausstellung 1889 in Paris wurde das Spiel „Die Türme von Hanoi" vorgestellt. Wenn der Scheibenturm auf eine der anderen Säulen umgeschichtet werden soll, gelten die folgenden Regeln:
- Die Scheiben dürfen nur einzeln von einer Säule zu einer anderen transportiert werden.
- Es darf niemals eine Scheibe mit einem größeren Durchmesser auf eine Scheibe mit einem kleineren Durchmesser gelegt werden.

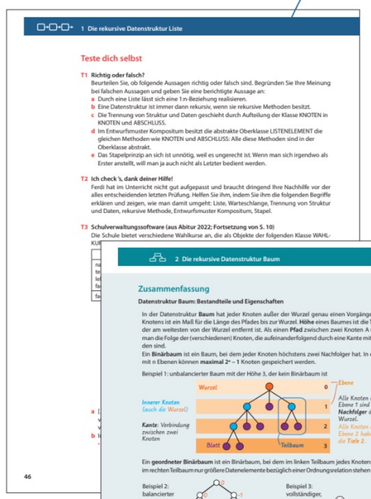
a Skizzieren Sie auf Papier eine Lösung für einen Anfangsturm mit zwei Scheiben und für einen mit den Scheiben. Geben Sie an, wie sich die Lösung für drei Scheiben auf die Lösung für zwei Scheiben zurückführen lässt.
b Geben Sie an, wie sich allgemein die Lösung für n Scheiben auf die Lösung für n-1 Scheiben zurückführen lässt.
c Der folgende rekursive Algorithmus löst das Problem für beliebige Turmhöhen. Die Methode TurmVerschieben bewegt die oberste Platte des Turms mit der Nummer „von" auf den Turm mit der Nummer „nach", die Türme haben die Nummern 0, 1 und 2.

```
methode TurmVerschieben(anzahlScheiben, von, nach)
    falls anzahlScheiben > 0
    dann
        frei = 3 - von - nach
        TurmVerschieben(anzahlScheiben - 1, von, frei)
        Bewegen(von, nach)
        TurmVerschieben(anzahlScheiben - 1, frei, nach)
    endefalls
endemethode
```

Erklären Sie die Funktionsweise unter Zuhilfenahme von b).
Erläutern Sie, was mit einem Stapel der Höhe 0 oder 1 passiert.
d Im Vorlagenprojekt sind bereits Methoden zur graphischen Darstellung des Verschiebevorgangs gegeben. Implementieren Sie die Methode TurmVerschieben und testen Sie den Algorithmus mit verschiedenen Höhen, aber nicht höher als 10 Scheiben.
e Für Schnelle: Bei Höhen über 10 dauert das Verschieben schon bald sehr lange. Zählen Sie für Türme der Höhen 1, 2 und 3, wie viele Verschiebungen insgesamt nötig sind. Erstellen Sie daraus eine Vermutung, wie viele Verschiebungen für einen Turm der Höhe n nötig sind, und begründen Sie damit die lange Laufzeit für hohe Türme.

93

„Teste dich selbst"-Aufgaben zum Überprüfen, ob Sie die wesentlichen Aspekte eines Hauptkapitels verstanden haben (Lösungen zur Selbstkontrolle am Ende des Buches!)

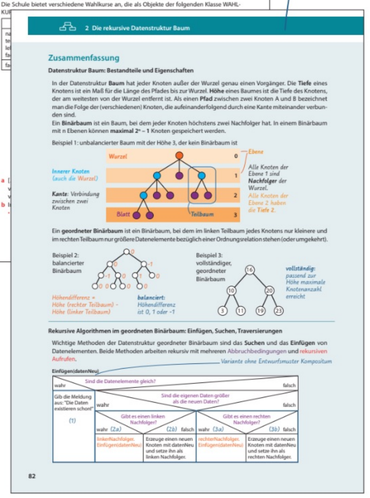

1 Die rekursive Datenstruktur Liste

Teste dich selbst

T1 Richtig oder falsch?
Beurteilen Sie, ob folgende Aussagen richtig oder falsch sind. Begründen Sie Ihre Meinung bei falschen Aussagen und geben Sie eine berichtigte Aussage an:
a Durch eine Liste lässt sich eine 1:n-Beziehung realisieren.
b Eine Datenstruktur ist immer dann rekursiv, wenn sie rekursive Methoden besitzt.
c Die Trennung von Struktur und Daten geschieht durch Aufteilung der Klasse KNOTEN in KNOTEN und ABSCHLUSS.
d Im Entwurfsmuster Kompositum besitzt die abstrakte Oberklasse LISTENELEMENT die gleichen Methoden wie KNOTEN und ABSCHLUSS: Alle diese Methoden sind in der Oberklasse abstrakt.
e Das Stapelprinzip an sich ist unnötig, weil es ungerecht ist. Wenn man sich irgendwo als Erster anstellt, will man ja auch nicht als Letzter bedient werden.

T2 Ich check 's, dank deiner Hilfe!
Ferdi hat im Unterricht nicht gut aufgepasst und braucht dringend Ihre Nachhilfe vor der alles entscheidenden letzten Prüfung. Helfen Sie ihm, indem Sie ihm die folgenden Begriffe erklären und zeigen, wie man damit umgeht: Liste, Warteschlange, Trennung von Struktur und Daten, rekursive Methode, Entwurfsmuster Kompositum, Stapel.

T3 Schulverwaltungssoftware (aus Abitur 2022; Fortsetzung von S. 10)
Die Schule bietet verschiedene Wahlkurse an, die als Objekte der folgenden Klasse WAHL-


2 Die rekursive Datenstruktur Baum

Zusammenfassung

Datenstruktur Baum: Bestandteile und Eigenschaften

In der Datenstruktur **Baum** hat jeder Knoten außer der Wurzel genau einen Vorgänger. Die **Tiefe** eines Knotens ist ein Maß für die Länge des Pfades bis zur Wurzel. **Höhe** eines Baumes ist die Tiefe des Knotens, der am weitesten von der Wurzel entfernt ist. Als einen **Pfad** zwischen zwei Knoten A und B bezeichnet man die Folge der (verschiedenen) Knoten, die aufeinanderfolgend durch eine Kante miteinander verbunden sind.
Ein **Binärbaum** ist ein Baum, bei dem jeder Knoten höchstens zwei Nachfolger hat. In einem Binärbaum mit n Ebenen können **maximal $2^n - 1$** Knoten gespeichert werden.

Beispiel 1: unbalancierter Baum mit der Höhe 3, der kein Binärbaum ist

Ein **geordneter Binärbaum** ist ein Binärbaum, bei dem im linken Teilbaum jedes Knotens nur kleinere und im rechten Teilbaum nur größere Datenelemente bezüglich einer Ordnungsrelation stehen (oder umgekehrt).

Beispiel 2: balancierter Binärbaum

Beispiel 3: vollständiger, geordneter Binärbaum

vollständig: passend zur Höhe maximale Knotenanzahl erreicht

Höhendifferenz = Höhe (rechter Teilbaum) - Höhe (linker Teilbaum)

balanciert: Höhendifferenz ist 0, 1 oder -1

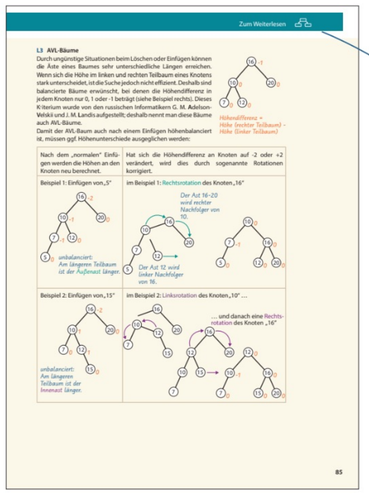
Rekursive Algorithmen im geordneten Binärbaum: Einfügen, Suchen, Traversierungen

Wichtige Methoden der Datenstruktur geordneter Binärbaum sind das **Suchen** und das **Einfügen** von Datenelementen. Beide Methoden arbeiten rekursiv mit mehreren Abbruchbedingungen und rekursiven Aufrufen.

Variante ohne Entwurfsmuster Kompositum

82

Kurze und prägnante Zusammenfassungen am Ende jedes Hauptkapitels und Merkkästen mit den zentralen Inhalten am Ende jedes Kapitels


Zum Weiterlesen

L3 AVL-Bäume

Durch ungünstige Situationen beim Löschen oder Einfügen können die Äste eines Baumes sehr unterschiedliche Längen erreichen. Wenn sich die Höhe im linken und rechten Teilbaum eines Knotens stark unterscheidet, ist die Suche jedoch nicht effizient. Deshalb sind balancierte Bäume erwünscht, bei denen die Höhendifferenz in jedem Knoten nur 0, 1 oder -1 beträgt (siehe Beispiel rechts). Dieses K-terium wurde von den russischen Informatikern G. M. Adelson-Velskii und J. M. Landis aufgestellt; deshalb nennt man diese Bäume auch AVL-Bäume.
Damit der AVL-Baum auch nach einem Einfügen höhenbalanciert ist, müssen ggf. Höhenunterschiede ausgeglichen werden:

Höhendifferenz = Höhe (rechter Teilbaum) - Höhe (linker Teilbaum)

Nach dem „normalen" Einfügen werden die Höhen an den Knoten neu berechnet.	Hat sich die Höhendifferenz an Knoten auf -2 oder +2 verändert, wird dies durch sogenannte Rotationen korrigiert.
Beispiel 1: Einfügen von „5"	im Beispiel 1: Rechtsrotation des Knoten „16"
Beispiel 2: Einfügen von „15"	im Beispiel 2: Linksrotation des Knoten „10" …

85

Texte Zum Weiterlesen, die Ihnen Einblicke in Berufe und Ausblicke auf inhaltlich angrenzende Anwendungsbereiche geben

Die Symbole beschreiben für jede Aufgabe Arbeitsweisen bzw. Zielsetzung.

Recherchieren, lesen, damit Sie selbst Verantwortung für den Lernfortschritt übernehmen können.

Vernetzen, damit Sie neue Inhalte mit anderen Bereichen und bereits Gelerntem in Beziehung setzen können.

Kommunizieren, weil es wichtig ist, anderen etwas erklären zu können und sich in der Gruppe zu besprechen.

Kooperieren, damit Ihre Produkte vielseitiger, hochwertiger und umfangreicher werden.

Kreativ arbeiten, damit Sie originell und mit persönlicher Note arbeiten können.

Analysieren, um Strukturen, Verfahren und Zusammenhänge zu erkennen und zu erfassen.

Modellieren, um Probleme zu verstehen und Lösungen zu planen.

Mit Rechnereinsatz lösen/implementieren, um mithilfe des Computers praktische Umsetzungen zu schaffen.

Handlungsorientiert lösen, damit Sie auch ohne Rechner aktiv werden können, z. B. bei Rollenspielen.

Reflektieren, begründen, damit Sie sich Lösungswege bewusst machen und ähnliche Aufgaben künftig schneller lösen können.

Offene Aufgabenstellung individuell ausgestalten, damit Sie sich selbst Ziele setzen und Ihren eigenen Lösungsweg suchen können.

* **Diese Inhalte und Aufgaben gehen über den Lehrplan hinaus.**

Das Debugging-Symbol markiert Aufgaben und Tipps zur Fehlersuche.

Auf informatikschulbuch.de finden Sie einführende Videos für die typischen Werkzeuge.

Werkzeugunabhängigkeit in Lehrtext und Aufgaben:

Basisprojekte für viele gängige Sprachen und typische Entwicklungsumgebungen

Java (BlueJ)

```
Java
synchronized … {
   try {wait();}
    catch (…){ }
    …
    notify();
}
```

Python

```
Python
import threading
lock = threading.Condition()
...
with lock:
        lock.wait()
        ...
        lock.notify()
```

Swift (Playgrounds)

```
Swift
let monitor = NSCondition()
...
monitor.lock
monitor.wait()
...
monitor.signal()
monitor.unlock
```

Wir wünschen Ihnen viel Spaß im Informatikunterricht mit diesem Buch!

Peter Brichzin Florian Janus Franz Jetzinger Johannes Neumeyer
Dr. Stefan Seegerer Klaus Reinold Albert Wiedemann

INFORMATIK 6

NTG | Grundlegendes Niveau

Listen | Bäume | Rekursion
Nebenläufigkeit und Threads
Informationssicherheit
Softwareprojekte

Redaktion: Dr. Ulf Rothkirch, Stina Tilsner, Martin Unger

Autoren: Peter Brichzin, Florian Janus, Franz Jetzinger, Johannes Neumeyer, Klaus Reinold, Dr. Stefan Seegerer, Albert Wiedemann
Technische Zeichnungen: Ingrid Schobel, Hannover
Illustrationen: Nicole Rademacher, Berlin; Natascha Welz
Umschlaggestaltung: Corinna Babylon, Berlin
Layout und technische Umsetzung: Reemers Publishing Services GmbH, Krefeld

www.cornelsen.de

1. Auflage, 1. Druck 2024

Alle Drucke dieser Auflage sind inhaltlich unverändert
und können im Unterricht nebeneinander verwendet werden.

© 2024 Cornelsen Verlag GmbH, Mecklenburgische Straße 53, 14197 Berlin

Das Werk und seine Teile sind urheberrechtlich geschützt.
Jede Nutzung in anderen als den gesetzlich zugelassenen Fällen bedarf
der vorherigen schriftlichen Einwilligung des Verlages.
Hinweis zu §§ 60 a, 60 b UrhG: Weder das Werk noch seine Teile dürfen ohne eine
solche Einwilligung an Schulen oder in Unterrichts- und Lehrmedien (§ 60 b Abs. 3 UrhG)
vervielfältigt, insbesondere kopiert oder eingescannt, verbreitet oder in ein Netzwerk eingestellt
oder sonst öffentlich zugänglich gemacht oder wiedergegeben werden.
Dies gilt auch für Intranets von Schulen und anderen Bildungseinrichtungen.
Der Anbieter behält sich eine Nutzung der Inhalte für Text und Data Mining im Sinne § 44 b UrhG
ausdrücklich vor.

Druck: Mohn Media Mohndruck, Gütersloh

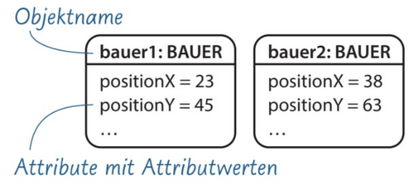
ISBN 978-3-637-02476-2
ISBN 978-3-637-02477-9 (E-Book)

PEFC-zertifiziert
Dieses Produkt stammt aus nachhaltig bewirtschafteten Wäldern und kontrollierten Quellen
PEFC/04-31-1033
www.pefc.de

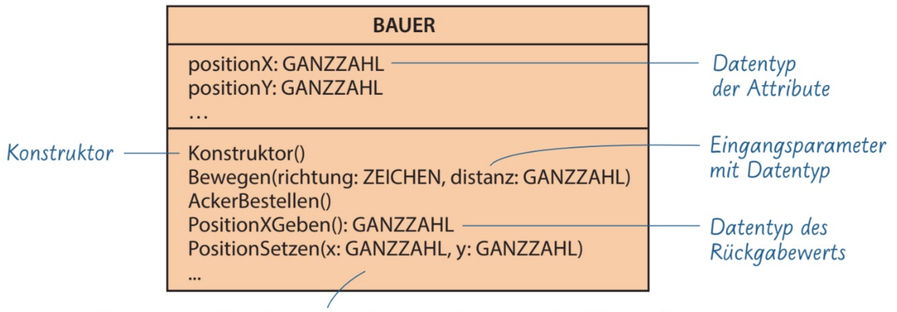
Startklar: Grundwissen und Grundkompetenzen

Im Folgenden werden wesentliche Begriffe der objektorientierten Modellierung und Programmierung zusammengefasst. Die Beispiele sind dem Kontext eines (fiktiven) Strategiespiels entnommen, dessen Ziel die Besiedelung eines neuen Planeten ist. Jeder Spieler hat verschiedene Spielfiguren (Minenarbeiter, Bauer, …) und Besitztümer (Minen, Ackerland, Schlüssel, …), mit denen er durch geschicktes Agieren Punkte sammeln kann.

Objekte und Klassen
Objekte haben **Attribute** mit **Attributwerten** und **Methoden**. Jedes Objekt besitzt einen eindeutigen Objektnamen und ist eine Instanz (Exemplar) genau einer **Klasse**. Von einer Klasse kann es beliebig viele Instanzen geben.

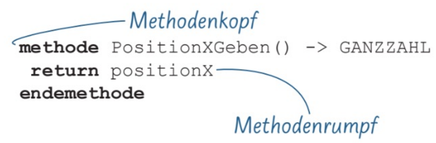


Objektkarten zweier Instanzen der Klasse BAUER



Klassenkarte der Klasse BAUER, gegenüber normalen Klassendiagrammen erweitert um Datentypen. Durch diese Erweiterung ist die Klassenkarte näher an der Implementierung. Im Buch sind solche Klassenkarten farblich hinterlegt, um sie von normalen Klassenkarten zu unterscheiden.

Im Quelltext einer Klasse werden die Attribute und Methoden genau beschrieben:
- **Deklaration** der Attribute: Festlegung der **Bezeichner,** je nach Sprache mit **Datentyp**
- **Deklaration** der Methoden:
  - Festlegung des Methodennamens, der Parameterliste und, falls vorhanden, des Datentyps des Rückgabewerts im Methodenkopf;
  - Festlegung der exakten Arbeitsweise eines Objekts bei einem Methodenaufruf über eine Anweisungsfolge im Methodenrumpf; diese Arbeitsweise wird nach außen nicht bekannt gegeben (Blackbox-Prinzip).



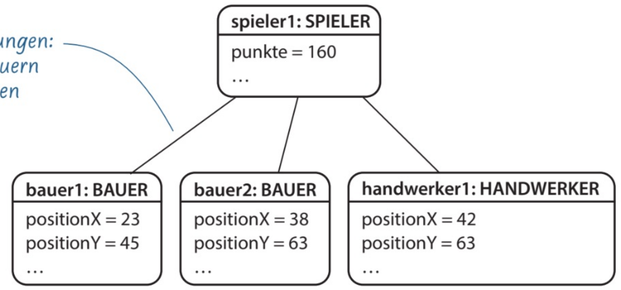
Der **Konstruktor** ist eine besondere Methode, die nach dem Erzeugen eines Objekts automatisch aufgerufen wird und dafür sorgt, dass das Objekt in einen definierten Anfangszustand versetzt wird.

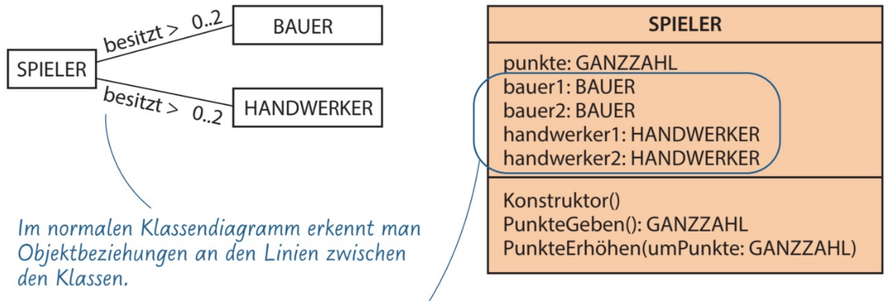
Je nach Programmiersprache gibt es verschiedene Konventionen, wie Methodennamen notiert werden. In diesem Buch beginnen wir mit einem Großbuchstaben. Selbstverständlich können Sie sich im Kurs auf eine andere Konvention einigen.

Objektbeziehungen durch Referenzattribute

Objektbeziehungen werden durch Referenzattribute realisiert. Diese Attribute speichern eine Referenz auf ein anderes Objekt. Der Wert `leer` (in Java: `null`, Python: `None`) eines Referenzattributs bedeutet, dass (vorübergehend) kein Objekt referenziert wird.

Objektdiagramm mit Beziehungen: spieler1 besitzt die beiden Bauern bauer1 und bauer2 sowie einen Handwerker handwerker1.





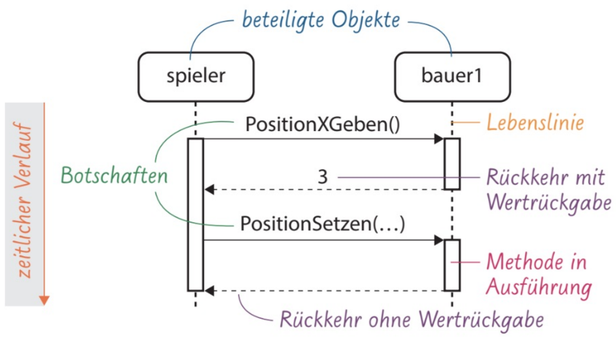
Im normalen Klassendiagramm erkennt man Objektbeziehungen an den Linien zwischen den Klassen.

In der erweiterten Form der Klassenkarte können die Referenzattribute bei den Attributen aufgeführt sein.
Genau wie bei einfachen Attributen erfolgt zunächst die Deklaration und danach die Initialisierung (erstmalige Zuweisung).

```
klasse SPIELER
    punkte: GANZZAHL
    bauer1: BAUER
    ...
    konstruktor()
        punkte = 0
        bauer1 = neu BAUER()
        ...
```

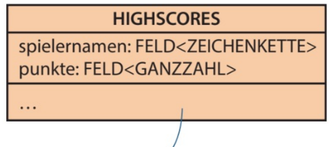
Objektkommunikation und Sequenzdiagramm

Mit ihren sichtbaren Methoden bieten Objekte nach außen **Schnittstellen** zur Kommunikation: Wird an ein Objekt ein Methodenaufruf passend zu einer Schnittstelle gesendet, so führt es die Methode aus.
Die Objektkommunikation ist nur dann möglich, wenn die Botschaft zielgerichtet zugestellt werden kann. Dafür besitzt das Sendeobjekt, ähnlich einem Adressbucheintrag, eine Referenz auf das Objekt, das die Botschaft empfängt. Die Kommunikation zwischen Objekten über Methodenaufrufe lässt sich mit **Sequenzdiagrammen** veranschaulichen.

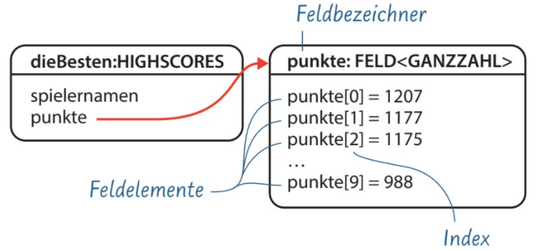


Felder

Die Datenstruktur **Feld** ermöglicht es, viele gleichartige Objekte effizient zu verwalten und so die 1:n-Beziehung umzusetzen. Hierfür gibt es hilfreiche Strukturen (Wiederholung „für alle“); das Feld bietet zudem hilfreiche Methoden (*Hinzufügen*, *Entfernen* …). Die **Feldelemente** werden beim Hinzufügen automatisch durchnummeriert. Über diese Nummer – **Index** genannt – kann auf die Feldelemente zugegriffen werden.

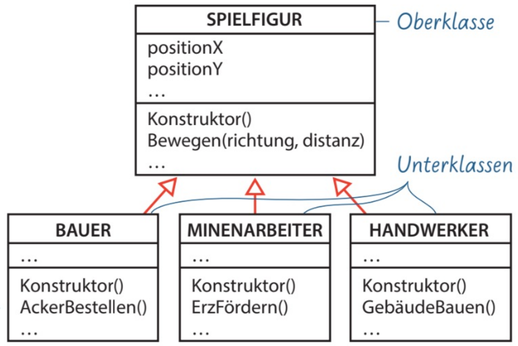


Klassenkarte, erweitert um Datentypen und Referenzattribute für zwei Felder



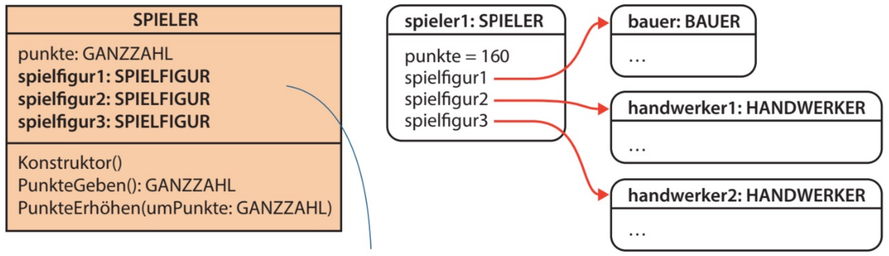
Vererbung und Polymorphismus

Mit dem Mechanismus der Vererbung lässt sich die Klassenbeziehung„ist ein“ zwischen **Unterklasse** (Spezialisierung) und **Oberklasse** (Generalisierung) umsetzen. Unterklassen erben alle Attribute und Methoden der Oberklasse, in ihnen können neue Attribute und Methoden hinzugefügt sowie bestehende Methoden neu definiert (überschrieben) werden.



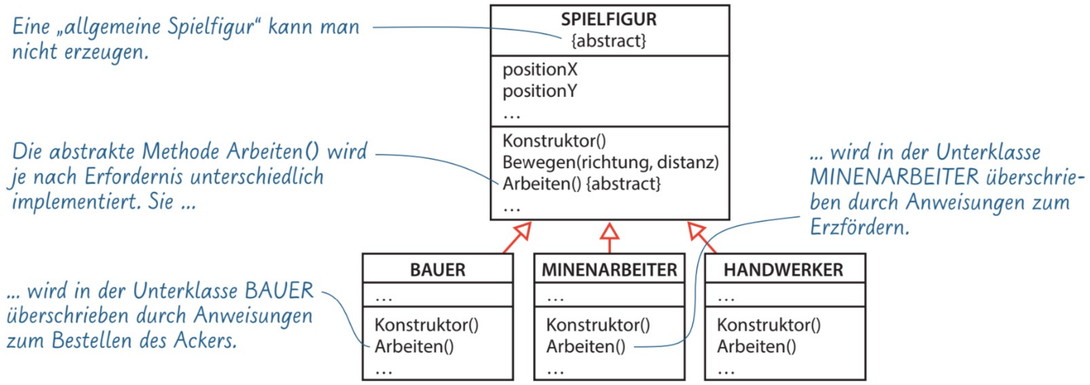
Die drei Unterklassen spezialisieren die Oberklasse SPIELFIGUR, indem jeweils die speziell relevanten Methoden AckerBestellen, ErzFördern bzw. GebäudeBauen hinzugefügt werden.

Polymorphismus bezeichnet die Eigenschaft, dass ein Referenzattribut nicht nur Objekte der angegebenen Klasse referenzieren kann, sondern auch Objekte aller dazu existierenden Unterklassen. Beim Zugriff über dieses Referenzattribut werden nicht die Methoden der Oberklasse aufgerufen, mit der das Referenzattribut definiert wurde, sondern die Methoden der Unterklasse, zu der das Objekt tatsächlich gehört.



In den Referenzattributen vom Typ SPIELFIGUR können auch Referenzen auf Objekte der Unterklassen BAUER, HANDWERKER und MINENARBEITER verwaltet werden.

Klassen werden als **abstrakte Klassen** bezeichnet, wenn von ihnen keine Objekte erzeugt werden sollen. In abstrakten Klassen können außer normalen Methoden auch Methoden vereinbart werden, deren Methodenrumpf dann in konkreten Unterklassen festgelegt werden muss. Solche Methoden werden **abstrakte Methoden** genannt.



Aufgaben

In den letzten Jahren haben Sie sich mit dem objektorientierten Modellieren und Programmieren vertraut gemacht. Im Wesentlichen beherrschen Sie in der erlernten Programmiersprache alles, was Sie bis zum Abitur brauchen! Mit diesen Kompetenzen sind Sie deshalb bereits in der Lage, Teile von Abituraufgaben zu lösen. Im Download-Angebot finden Sie einen „Spickzettel" mit den wichtigsten Befehlen in der von Ihnen verwendeten Programmiersprache und außerdem eine Lösung zu den Aufgaben, falls Sie an einzelnen Stellen nicht weiterkommen.

1 Schulverwaltungssoftware (aus Abitur 2022)

Für die John-von-Neumann-Realschule soll eine Schulverwaltungssoftware entwickelt werden. Es sollen die Daten von allen Schülerinnen und Schülern, Lehrkräften, Schulklassen, Erziehungsberechtigten und Räumen verwaltet werden. Von jeder Schülerin und jedem Schüler soll der Name, das Geburtsdatum und die Schulklasse gespeichert werden. Diese wird anhand ihrer Bezeichnung (z. B. 7a) identifiziert. Von Lehrkräften soll der Name und das eindeutige Kürzel registriert werden. Ebenso soll für Schulklassen gespeichert werden, welche Lehrkräfte sie unterrichten. Von Räumen soll eine eindeutige Nummer sowie die Anzahl der Sitzplätze erfasst werden. Jeder Schulklasse ist genau ein Raum als Klassenraum und jeder Schülerin bzw. jedem Schüler mindestens eine erziehungsberechtigte Person zugeordnet, deren Name erfasst wird.

a Erstellen Sie ein zur beschriebenen Situation passendes Klassendiagramm. Auf die Angabe von Methoden und nicht im Text genannten Attributen kann verzichtet werden.

b Erläutern Sie im Sachzusammenhang ein mögliches Szenario, für das man das Prinzip der Spezialisierung für eine der Klassen Ihres Diagramms aus Teilaufgabe 1a sinnvoll anwenden kann. Stellen Sie diese Vererbungsbeziehung in einem Klassendiagramm dar.

Weiterführende Aufgaben, nicht mehr Teil der Abituraufgabe:

c Implementieren Sie von der in b) beschriebenen Vererbungsbeziehung die Oberklasse sowie zwei Unterklassen. Jede der Klassen sollte zumindest ein eigenes Attribut haben. Ergänzen Sie jeweils auch den Konstruktor, Standardmethoden zum Setzen und Geben der Attributwerte sowie eine Methode *InformationAusgeben*, die alle Werte eines Objekts auf der Konsole ausgibt.

d Mehrere Objekte der Unterklassen aus Teilaufgabe c) sollen gemeinsam in einer eindimensional indizierten Datenstruktur (Feld oder Liste) verwaltet werden. Erstellen Sie eine Klasse SCHULVERWALTUNG mit einem solchen Feld. Über Methodenaufrufe soll es möglich sein, Objekte in das Feld einzufügen.

e Erklären Sie knapp den Begriff Polymorphismus und das „Überschreiben einer Methode" am Beispiel aus d).

f Regelmäßiges Testen ist in der Softwareentwicklung wichtig. Erstellen Sie z. B. mithilfe einer Testklasse ein Szenario mit mindestens fünf Objekten, das Sie schnell wiederherstellen können.

g Ergänzen Sie in der Klasse SCHULVERWALTUNG eine Methode, mit der die Informationen aller in der Liste gespeicherten Objekte ausgegeben werden können.

h Vertiefungsmöglichkeit: Entwickeln Sie die Schulverwaltungssoftware in einem kleinen Übungsprojekt arbeitsteilig weiter.

2 Burning Man (aus Abitur 2021)

Im US-Bundesstaat Nevada findet jedes Jahr ein neuntägiges Kunstfestival mit ca. 70 000 Besuchern statt, dessen Höhepunkt das Verbrennen einer überdimensionalen Statue – des Burning Man – am sechsten Festivaltag ist. Es wird angenommen, dass für das Festival eine zentrale Verwaltungssoftware eingesetzt wird. Mit dieser Software kommuniziert eine App, über die sich die Besucher des Festivals mit ihrem Namen, ihrem Geburtsdatum und ihrer E-Mail-Adresse registrieren müssen. Die Besucher übernachten alle in Camps auf dem Festivalgelände. Nach dem Registrieren muss jeder Besucher über die App eines dieser Camps zum Übernachten auswählen. Mithilfe der App können auch die Veranstaltungen des Festivals angezeigt und gebucht werden. Für jede Veranstaltung ist eine maximale Besucherzahl festgelegt. Außerdem werden auf dem Festival viele Kunstwerke ausgestellt, die einen eindeutigen Namen haben und deren jeweiliger Künstler bekannt ist. Die Besucher können Kunstwerke in der App mit „Gefällt mir" markieren. In der eigens für das Festival konzipierten „Stadt" Black Rock City gibt es mehrere festgelegte Orte.

Jedem Camp, jeder Veranstaltung und jedem Kunstwerk ist genau ein Ort zugeordnet. An jedem Ort befindet sich höchstens ein Camp. Dagegen kann es an einem Ort mehrere Veranstaltungen und Kunstwerke geben.

a Entwickeln Sie für die beschriebene Situation ein Klassendiagramm, das ausschnittsweise als Planungsgrundlage für die Verwaltungssoftware des Festivals dienen kann. Verwenden Sie die Klassen BESUCHER, CAMP, VERANSTALTUNG, KUNSTWERK und ORT. Auf die Angabe von Methoden kann verzichtet werden.

Zur Verwaltung der Plätze eines Camps hat die Klasse CAMP als Attribut ein Feld besucher, in dem Referenzen auf BESUCHER-Objekte gespeichert werden können. Die Länge des Felds entspricht dabei genau der Anzahl der Plätze des Camps und der Feldindex der jeweiligen Platznummer.

b Nennen Sie zwei Kriterien, die in diesem Kontext zur Entscheidung für die Datenstruktur Feld geführt haben könnten.

c Die Klasse CAMP soll eine Methode *nachbarplaetzeVerfuegbar(n)* besitzen, die überprüft, ob noch mindestens n benachbarte Plätze frei sind, und den entsprechenden Wahrheitswert zurückgibt. Notieren Sie […] eine Implementierung der Methode *nachbarplaetzeVerfuegbar*. Dabei kann vereinfachend davon ausgegangen werden, dass benachbarte Feldelemente im Feld besucher Plätzen entsprechen, die auch in der Realität benachbart sind.

3 Bestellsystem

Im Restaurant „Knödel-Karli“ soll ein elektronisches Bestellsystem eingeführt werden. In Zweierteams unterstützen Sie die Einführung des Systems.

- **a** Im beigefügten Projekt existieren noch einige Syntaxfehler. Finden und beseitigen Sie diese.
- **b** Erstellen Sie ein Klassendiagramm mit Beziehungen, Attributen und Methoden, um einen Überblick über das Projekt zu gewinnen. Besprechen Sie dabei mündlich, wie die Klassen und Methoden zusammenwirken.
- **c** Erstellen Sie eine Testklasse für das Projekt für ein Restaurant mit 10 Tischen. Für verschiedene Tische sollen Bestellungen aufgenommen werden. Für die Tische soll am Ende die Rechnung ausgegeben werden.
  Testen Sie das Projekt anhand der Testklasse. Beseitigen Sie auftretende Fehler.
- **d** Kunden, die einen zuvor bereits einmal vergebenen Tisch nutzen, beschweren sich, dass auch die Posten der Vorgänger weiterhin auf der Rechnung stehen. Beheben Sie diesen Fehler: Nach Ausgabe der Rechnung sollen alle Posten des Tisches gelöscht werden.
- **e** Erstellen Sie in der Klasse BESTELLSYSTEM eine Methode zur Anzeige der Speisekarte.
- **f** Für Schnelle: Überlegen Sie sich weitere Vorschläge zur Verbesserung des Systems. Wählen Sie einen Vorschlag aus und setzen Sie diesen um.

4 Richtig oder falsch?

Beurteilen Sie, ob folgende Aussagen richtig oder falsch sind. Begründen Sie Ihre Meinung bei falschen Aussagen und geben Sie eine berichtigte Aussage an:

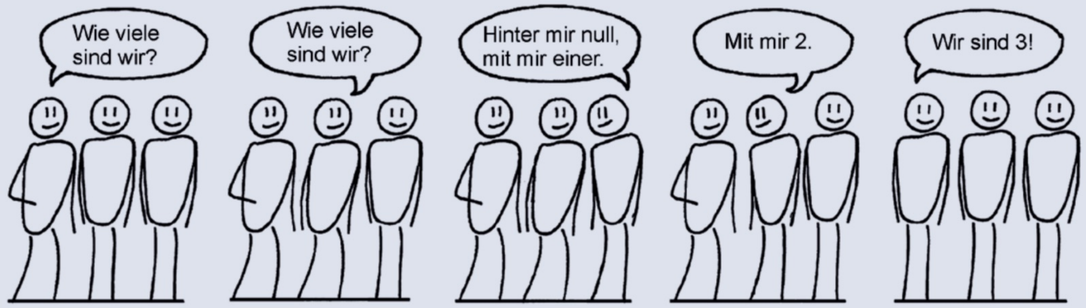
- **a** Eine Unterklasse erbt nur die Attribute der Oberklasse.
- **b** Im Konstruktor werden für die Attribute sinnvolle Startwerte festgelegt.
- **c** Eine integrierte Entwicklungsumgebung ist eine Programmiersprache, mit der Datenbanken bearbeitet werden können.
- **d** Mit den logischen Funktionen UND sowie ODER können Bedingungen formuliert werden.
- **e** In einer Oberklasse kann eine Methode der Unterklasse überschrieben werden.
- **f** Die beiden Bedingungen *IstSichtbar()* == WAHR und *IstSichtbar()* sind gleichwertig.
- **g** Ein Referenzattribut kann man sich vorstellen wie eine Schachtel, in der ein Wert gespeichert ist.
- **h** Nach dem Kapselungsprinzip sollte der direkte Zugriff auf Attribute und Methoden verhindert werden.
- **i** Abstrakte Methoden können nur in abstrakten Klassen vorkommen.

1 Die rekursive Datenstruktur Liste

In diesem Kapitel erfahren Sie, ...

... wie die Liste als rekursive Datenstruktur umgesetzt werden kann.

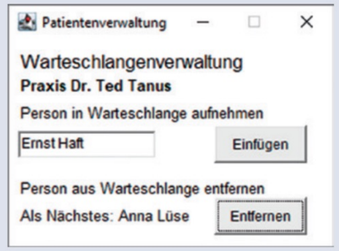
... wie rekursive Methoden helfen, mit der Liste zu arbeiten.



... wie aus Prototypen ein immer besseres Produkt entsteht.

1.1 Prototyp 1: Die rekursive Warteschlange

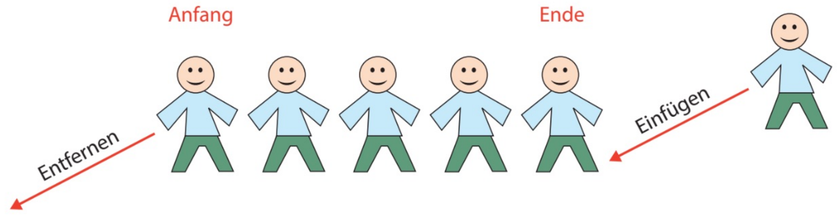
In einer Arztpraxis soll die Verwaltung der wartenden Patienten mit Hilfe eines Programms neu gestaltet werden.



a Erproben und analysieren Sie, wie das Problem im Projekt „Lösung mit Feld“ gelöst wurde.
b Beschreiben Sie, wie in der „Lösung ohne Feld“ die 1:n-Beziehung zwischen Praxis und Patienten umgesetzt ist.
c Analysieren Sie das Einfügen am Ende der Warteschlange in der „Lösung ohne Feld“ und begründen Sie, weshalb die Lösung ohne Feld wesentlich kürzer ist.
d Entwickeln Sie für die Lösung ohne Feld eine Methode zum Entfernen des ersten Patienten aus der Warteschlange.

Prinzip der Warteschlange

Die Datenstruktur **Warteschlange** (auch: Schlange, Queue, Puffer) begegnet uns im Alltag an vielen Stellen, ob beim Arzt im Wartezimmer, beim Anruf in einem Callcenter, bei der Playlist eines Audioplayers oder als Warteschlange eines Druckers. Neben dem Feld bietet sie eine weitere Möglichkeit zur Realisierung einer 1:n-Beziehung.



→FIFO, Abkürzung für engl. first in, first out

In der Warteschlange werden die Elemente nach dem sogenannten →FIFO-Prinzip verwaltet:
- Neue Elemente werden am Ende der Warteschlange eingefügt.
- Entfernt wird immer das Element ganz vorne in der Warteschlange.

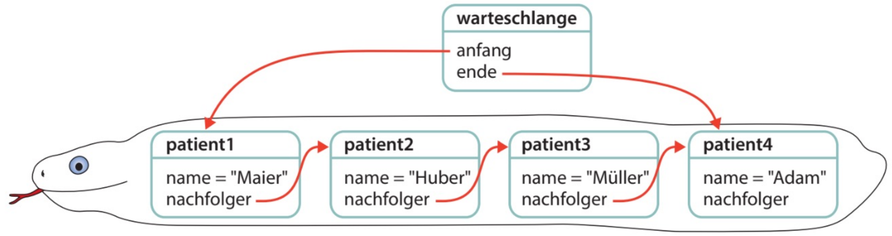
Für diese beiden Anforderungen benötigt die Warteschlange jeweils eine Methode *Einfügen(patientNeu)* und *Entfernen() -> PATIENT*.
Darüber hinaus gibt es weitere wichtige Forderungen, die eine Warteschlange erfüllen soll:
- Die Anzahl der verwalteten Elemente soll prinzipiell unbeschränkt sein.
- Die Anzahl der reservierten, aber unbesetzten Speicherplätze soll möglichst gering sein. Optimal ist es, wenn es keinen unbesetzten Platz gibt.

Um die Reihenfolge der Elemente abzubilden, müssen die Elemente geeignet strukturiert werden. Die Ordnung kann dadurch hergestellt werden, dass jedes Element seinen Nachfolger kennt. Die Beziehung zwischen einem Element und seinem Nachfolger bleibt auch dann erhalten, wenn weiter vorne ein Patient die Warteschlange verlässt. Im Unterschied zur Verwendung eines Feldes lassen sich hier aufwendige Methoden zum Aufrücken vermeiden.

Modellierung der Warteschlange

In der Arztpraxis besteht die Warteschlange aus Patienten. Jeder Patient trägt seine spezifischen Informationen. Im Beispiel soll dies nur der Name sein; in der Realität werden hier viele weitere Patientendaten wie die Krankenkasse oder gespeicherte Behandlungsdaten hinzukommen.

Jeder Patient „merkt“ sich außerdem seinen Nachfolger. Die Arzthelfer greifen über die Warteschlange nur auf den ersten und den letzten Patienten zu, um Patienten in den Wartebereich einzufügen und von dort zu holen.
Im Objektdiagramm lassen sich diese Zusammenhänge folgendermaßen veranschaulichen:

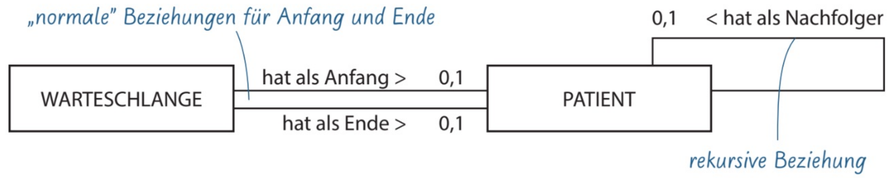


Bei eindeutigen Datentypen kann man auf deren Angabe im Objektdiagramm verzichten.

Eine rekursive Beziehung

Die Warteschlange hat zwei Beziehungen zu Patienten: Sie verwaltet eine Referenz auf den ersten und auf den letzten Patienten. Ein Patient hat als Nachfolger keinen oder einen Patienten. Zwischen zwei Patienten gibt es also die „hat als Nachfolger“-Beziehung. Eine solche Beziehung zurück zu einem Objekt der gleichen Klasse nennt man →**rekursive Beziehung**. Auf diese Weise kann eine beliebige und prinzipiell unbeschränkte Anzahl an Patienten verwaltet werden. Die Warteschlange ist eine **rekursive Datenstruktur**.

→lat. recurrere: zurückgehen



Zur Umsetzung der Beziehungen benötigt die Klasse WARTESCHLANGE zwei Referenzattribute anfang und ende vom Typ PATIENT. In der Klasse PATIENT benötigt man ein Referenzattribut nachfolger zur Umsetzung der rekursiven Beziehung. Das Klassendiagramm, in dem zusätzlich die Referenzattribute und die Datentypen mit angegeben sind, hat die folgende Form:

WARTESCHLANGE
anfang: PATIENT ende: PATIENT
Konstruktor() Einfügen(PATIENT) Entfernen(): PATIENT

PATIENT
name: ZEICHENKETTE nachfolger: PATIENT
Konstruktor(ZEICHENKETTE) NachfolgerSetzen(PATIENT) NachfolgerGeben(): PATIENT NameGeben(): ZEICHENKETTE

Klassenkarten erweitert um Datentypen und Referenzattribute

Die Datenstruktur **Warteschlange** stellt eine Möglichkeit zur Realisierung einer 1:n-Beziehung mit unbeschränktem Wert für n dar. Bei einer Warteschlange ist das Einfügen von Elementen ausschließlich am Ende und das Entfernen nur am Anfang möglich.
Eine Beziehung zwischen Objekten der gleichen Klasse nennt man **rekursive Beziehung**.
Zwischen den Elementen der Warteschlange besteht die rekursive Beziehung „hat als Nachfolger“. Über das Referenzattribut nachfolger werden die Objekte miteinander verkettet.

Aufgaben

1 Einfach oder kompliziert?
Julia und Gianna haben für das Einfügen und das Entfernen in die Warteschlange der Arztpraxis jeweils unterschiedliche Methoden in Pseudocode entworfen. Begründen Sie, welche Fälle bei der kürzeren Version jeweils nicht berücksichtigt werden, indem Sie ein Objektdiagramm vor und nach dem Methodenaufruf zeichnen. Erläutern Sie auch, weshalb die längere Version korrekt arbeitet.

a

Klasse WARTESCHLANGE
Methode Einfügen(patientNeu)

```
ende.NachfolgerSetzen(patientNeu)
ende = patientNeu
```

Klasse WARTESCHLANGE
Methode Einfügen(patientNeu)

```
falls ende != leer dann
 ende.NachfolgerSetzen(patientNeu)
sonst
 anfang = patientNeu
endefalls
ende = patientNeu
```

b

Klasse WARTESCHLANGE
Methode Entfernen() -> PATIENT

```
p: PATIENT
p = anfang
anfang = anfang.NachfolgerGeben()
return p
```

Klasse WARTESCHLANGE
Methode Entfernen() -> PATIENT

```
p: PATIENT
p = anfang
falls anfang != leer dann
 anfang = anfang.NachfolgerGeben()
 falls anfang == leer dann
  ende = leer
 endefalls
endefalls
return p
```

2 Diashow/Audioplayer/Videoplayer
Mit der Klasse BILDANZEIGE in der Vorlage kann man ein Bild anzeigen, das über seinen Namen identifiziert ist und im Unterordner „Bilder" des Projekts liegt. Ziel ist die Entwicklung eines Programms, das eine Folge vorher festgelegter Bilder nacheinander anzeigt (Analog könnte ein Audio- oder Videoplayer entwickelt werden).

a Entwickeln Sie eine Klasse BILD, die den Namen mit Pfad des Bildes verwaltet und analog zur Klasse PATIENT aus dem Lehrtext eine Referenz auf das nachfolgend anzuzeigende Bild verwaltet.

b Entwickeln Sie eine Klasse WARTESCHLANGE, die das Einfügen und Entfernen der Bilder analog zum Beispiel aus dem Lehrtext für Bilder erlaubt. Wird ein Bild aus der Warteschlange entfernt, so soll es angezeigt werden.

3 Anwendungen der Warteschlange
Beschreiben Sie neben den im Lehrtext erwähnten Beispielen weitere Anwendungen, die das Prinzip der Warteschlange nutzen.

4 Feld oder Warteschlange – was ist besser?

a Die Patientenverwaltung in einer Arztpraxis kann über ein Feld oder über eine Schlange erfolgen. Erörtern Sie Vor- und Nachteile der beiden Lösungsansätze.

Entscheiden und begründen Sie bei den folgenden Teilaufgaben, ob ein Feld oder eine Schlange für die Problemlösung geeignet ist.

b System zur Festlegung der Startreihenfolge bei einem Ski-Weltcuplauf

c System zur Erfassung der Ergebnisse bei einem Ski-Weltcuplauf

d System zur Aufnahme der Ergebnisse bei einem Marathonlauf

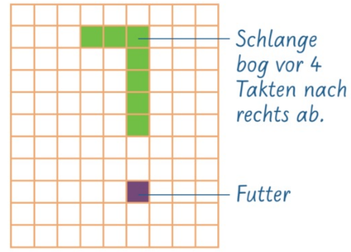
e Ablaufplan für ein Tischtennisturnier nach dem K.-o.-Prinzip: Es soll angezeigt werden, welche zwei Spieler als nächstes gegeneinander antreten. Der Gewinner gelangt in die nächste Runde.

5 Snake – Level 1 (Projektmöglichkeit)
Snake ist ein einfaches Computerspiel, das um 1978 entwickelt wurde. In einer Gitterwelt bewegt sich eine Schlange und sucht das nächste Futterstück. Findet sie eines, so wird sie am Ende um ein Element länger und ein neues Futterstück wird gesetzt. Das Spiel endet, wenn der Kopf auf den Körper der Schlange oder auf den Rand trifft.

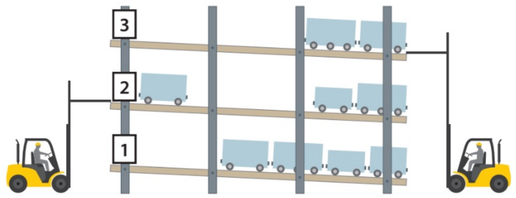
a Analysieren Sie in der Klasse SCHLANGE, wie die 1:n-Beziehung für den Rumpf über ein Feld realisiert ist.

b Realisieren Sie diese 1:n-Beziehung stattdessen unter Verwendung der Klasse WARTESCHLANGE, die Objekte der Klasse RUMPFSYMBOL verwaltet. Auch die Vorgänger-Nachfolger-Beziehung für Rumpfsymbole muss dabei umgesetzt werden.

c Begründen Sie, ob eine Highscore-Liste über die Datenstruktur Schlange sinnvoll verwaltet werden kann.



6 Hochregallager (aus Abitur 2013)
Die effiziente Lagerung von Waren erfolgt in großen Lagerzentren oft in Hochregallagern. Wird ein Regal prinzipiell immer von der gleichen Seite mit Waren bestückt, während die Entnahme der vorher eingelagerten Waren stets von der anderen Seite erfolgt, spricht man von einem Durchlaufregal. Das in dieser Aufgabe betrachtete Durchlaufregal besteht aus drei übereinander liegenden, zehn Meter langen Fächern mit geneigten Böden. Darin werden rollfähige Container unterschiedlicher Länge gelagert. Die Fächer sind – beginnend mit 1 – von unten nach oben durchnummeriert.



In Produktbeschreibungen von Hochregallager-Herstellern kann man lesen: „Die Lagerung in Durchlaufregalen erfolgt nach dem FIFO-Prinzip. Dieses für die Warenlagerung grundlegende Prinzip geht davon aus, dass Artikel, die zuerst eingelagert werden, auch als erste verbraucht bzw. weiterverarbeitet werden (First In – First Out)."

a Geben Sie eine spezielle Datenstruktur an, die sich zur Implementierung eines Faches nach dem FIFO-Prinzip eignet, und beschreiben Sie kurz die grundlegenden Methoden dieser Struktur. Hinweis: Eine Implementierung ist nicht notwendig.

b Erläutern Sie an einem konkreten Beispiel, warum das FIFO-Prinzip für das Hochregal als Ganzes nicht zwingend gelten muss.

c Eine andere Datenstruktur ist der Stapel. Beschreiben Sie eine Lagervariante, die mithilfe eines Stapels modelliert werden kann.

Wie aus der Alltagssprache bekannt, ist der Stapel eine Struktur, bei der das zuletzt eingefügte Element als erstes wieder entnommen wird.

1.2 Prototyp 2: Trennung von Struktur und Daten

a Analysieren und erproben Sie im Team das gegebene Projekt zur Verwaltung eines Handballspiels.
b Begründen Sie, weshalb das gegebene Projekt unübersichtlich und das Ergänzen von Funktionalitäten an unterschiedlichen Stellen durch mehrere Entwickler schwierig ist.
c Skizzieren Sie einen Vorschlag für ein Klassendiagramm zur besseren Strukturierung der Situation.

Verschiedene Aufgaben, verschiedene Klassen

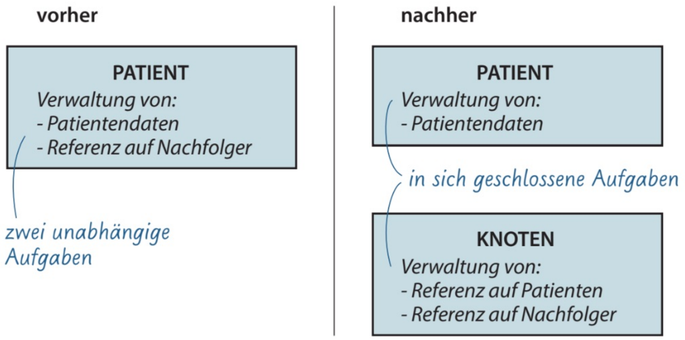
Im Prototyp 1 der Warteschlange muss sich ein Objekt der Klasse PATIENT momentan um zwei verschiedene Aufgaben kümmern: Einerseits verwaltet es die Referenz auf den Nachfolger, andererseits die Daten eines Patienten, die im Beispiel zur Vereinfachung nur aus dem Namen des Patienten bestehen, in der Realität aber viel komplexer sind (Patientendaten, Krankheiten etc.). Während aber die Daten des Patienten auf längere Zeit in der Arztpraxis gespeichert werden sollten, verliert die Referenz auf den Nachfolger ihre Bedeutung, sobald der Patient das Wartezimmer verlässt.

Ein Ziel der objektorientierten Programmierung ist es, komplexe Systeme in einfachere Bestandteile zu zerlegen, indem jede Klasse für einen **abgegrenzten Aufgabenbereich** zuständig ist. Die Trennung in verschiedene Aufgabenbereiche schafft nicht nur Übersichtlichkeit und macht das Projekt dadurch leichter **wartbar**, sondern begünstigt auch die **Wiederverwendbarkeit**; so lässt sich etwa die Grundstruktur der Warteschlange auf viele ähnliche Situationen übertragen, von denen die Patientenverwaltung nur ein Spezialfall ist.

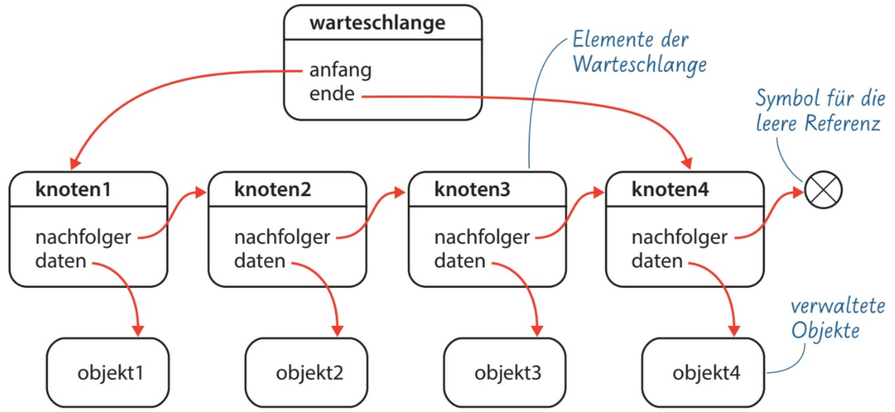
Für die Warteschlange der Arztpraxis soll deshalb zwischen zwei Klassen differenziert werden:

→„Knoten“ im Sinne eines Verbindungspunktes

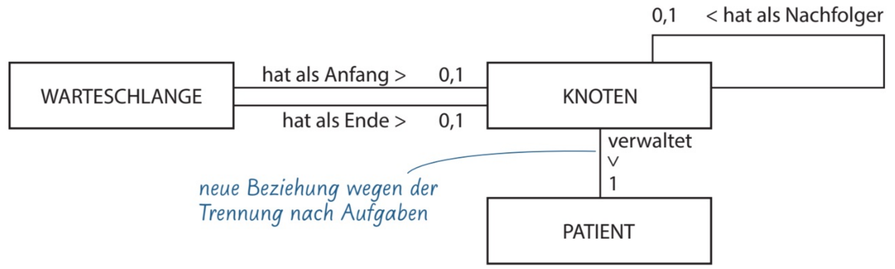
- eine Klasse →KNOTEN, die eine Referenz auf den Nachfolger sowie eine Referenz auf den zugehörigen Patienten verwaltet
- eine Klasse DATENELEMENT, die die eigentlichen Objekte verwaltet (hier: PATIENT).



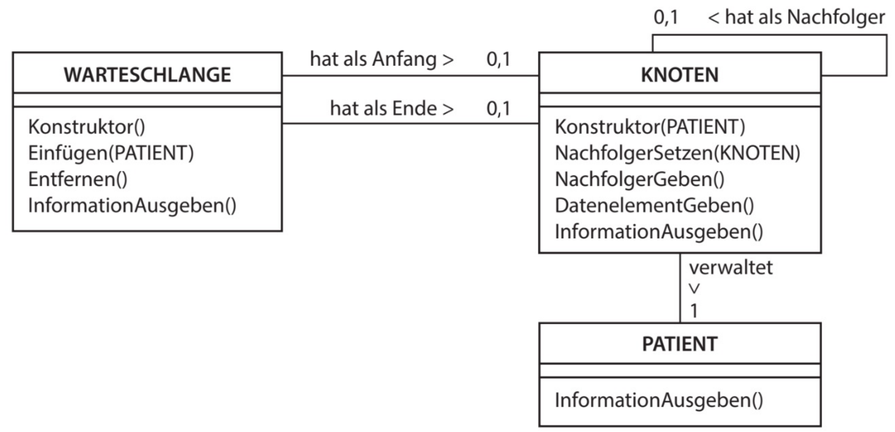
Die Warteschlange referenziert dann nur den ersten und den letzten Knoten.



Aus diesen Überlegungen ergibt sich folgendes Klassendiagramm für die Beziehungen zwischen den Klassen:



Auch die Methoden müssen dann auf Knoten und Patienten aufgeteilt werden: Der Knoten erhält die Methoden, die für die Verwaltung der Struktur notwendig sind (*NachfolgerSetzen*, *NachfolgerGeben* sowie die zusätzliche Methode *DatenelementGeben*, um das Datenelement (hier: den Patienten) zurückzugeben). Der Patient erhält die Methoden zur Verwaltung der Patientendaten (im vereinfachten Beispiel eine Methode *InformationAusgeben* zur Ausgabe des Namens).

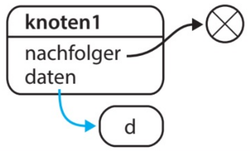


Die Methoden des Knotens und der Warteschlange werden im Folgenden beschrieben:

Klasse KNOTEN
Konstruktor(d: PATIENT)

Erhält eine Referenz auf ein Objekt der Klasse PATIENT und erzeugt ein Warteschlangenelement, das diese Referenz im Attribut daten und eine leere Referenz im Attribut nachfolger speichert.

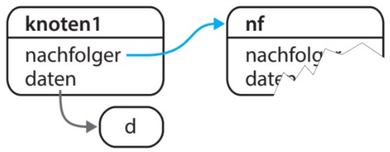
```
nachfolger = leer
daten = d
```



NachfolgerSetzen(nf: KNOTEN)

Setzt den Wert des Referenzattributs nachfolger auf den übergebenen Wert.

```
nachfolger = nf
```



NachfolgerGeben() -> KNOTEN

Gibt die Referenz auf den Nachfolger zurück.

DatenelementGeben() -> PATIENT

Gibt die Referenz auf das Datenelement (hier: Patient) zurück.

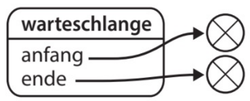
InformationAusgeben()

Ruft die Methode *InformationAusgeben* für das Referenzattribut daten auf.

Klasse WARTESCHLANGE
Konstruktor()

Erzeugt eine leere Warteschlange.

```
anfang = leer
ende = leer
```

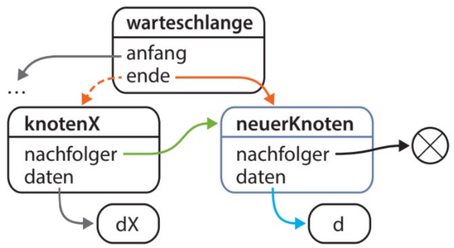


Einfügen(d: PATIENT)

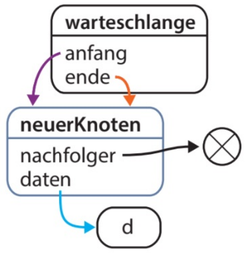
Reiht ein Objekt am Ende der Warteschlange ein.

```
neuerKnoten: KNOTEN
neuerKnoten = neu KNOTEN(d)
falls ende != leer dann
 ende.NachfolgerSetzen(neuerKnoten)
sonst
 anfang = neuerKnoten
endefalls
ende = neuerKnoten
```

Warteschlange zuvor nicht leer:



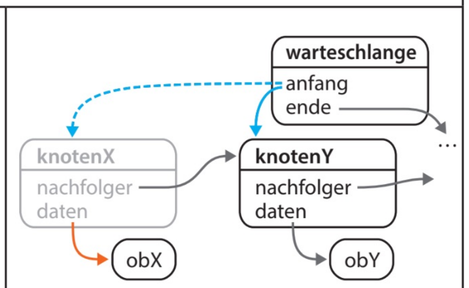
Warteschlange zuvor leer:



Entfernen() -> PATIENT

Entfernt den ersten Knoten aus der Warteschlange und gibt das zugehörige Datenelement (hier: PATIENT) zurück.

```
ausgabe: PATIENT
ausgabe = leer
falls anfang != leer dann
 ausgabe=anfang.DatenelementGeben()
 anfang=anfang.NachfolgerGeben()
 falls anfang == leer dann
  ende = leer
 endefalls
endefalls
return ausgabe
```



InformationAusgeben()

Ruft die Methode *InformationAusgeben* für den ersten Knoten auf.

Jede Klasse sollte zur besseren **Wartbarkeit** und **Wiederverwendbarkeit** nach Möglichkeit nur einen **Aufgabenbereich** haben. Aus diesem Grund wird bei der Warteschlange eine Klasse KNOTEN eingeführt, die nur die Referenzen auf ein Datenelement und den Nachfolgerknoten verwaltet.
Die **Daten** werden in einer eigenen Klasse verwaltet.

Aufgaben

1 Rollenspiel

Im Rollenspiel lassen sich gut die Struktur und die Kommunikation der Objekte analysieren und besser verstehen.

- **a** Bilden Sie eine Warteschlange, die drei Patienten verwaltet und die Trennung von Struktur und Daten berücksichtigt. Zur Darstellung der Referenzen kann man selbstgebastelte Schnüre mit Pfeilspitzen verwenden oder den referenzierten Objekten eine Hand auf die Schulter legen.
- **b** Erproben Sie das Einfügen eines neuen Patienten in die Warteschlange der Arztpraxis. Die Schülerinnen und Schüler, die nicht aktiv am Rollenspiel teilnehmen, beobachten, ob das Spiel der Beschreibung im Lehrtext entspricht. Diskutieren Sie die Unterschiede zum Lehrtext und verbessern Sie das Spiel in weiteren Durchläufen.
- **c** Verfahren Sie ebenso beim Entfernen eines Patienten.
- **d** Diskutieren Sie, wie das Rollenspiel ohne Trennung von Struktur und Daten ablaufen würde.
- **e** Erproben Sie ein Rollenspiel für eine Arztpraxis mit „echten Menschen". Formulieren Sie Gemeinsamkeiten und Unterschiede zur objektorientierten Umsetzung.

2 Umsetzung einer Warteschlange

- **a** Planen Sie, wie Sie das Projekt aus Kapitel 1.1 möglichst effizient umbauen können, ohne den Code komplett neu schreiben zu müssen, um die Trennung von Struktur und Daten zu realisieren. Setzen Sie die Planung um.
- **b** Testen Sie die Methoden der Warteschlange in verschiedenen Anwendungssituationen (Warteschlange zuvor leer; ein Patient in der Warteschlange; mehrere Patienten in der Warteschlange). Verbessern Sie das Programm bei Bedarf.

3 Trennung nach Aufgabenbereichen

Für die Datenbank einer Schulbibliothek wird eine Tabelle Entleihvorgang erstellt mit den Spalten buchtitel, buchautor, verlag, jahr, autor1, autor2, entleihername, entleiherklasse, entleihdatum, rückgabedatum.

- **a** Benennen Sie Gründe, die gegen eine solche Umsetzung sprechen. Geben Sie verschiedene Aufgaben der Tabelle an.
- **b** Erstellen Sie ein Klassendiagramm, das sorgfältig auf eine Trennung nach Aufgabenbereichen achtet.

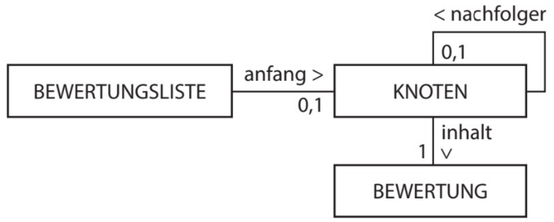
4 Analyse alter Projekte

- **a** Analysieren Sie arbeitsteilig anhand von Projekten aus dem Vorjahr, wie Sie die Trennung nach Aufgabengebieten bei der Erstellung Ihrer Klassen praktiziert haben.
- **b** Überprüfen Sie, inwieweit dort bereits eine Trennung von Struktur und Daten stattgefunden hat, wenn im Projekt die Datenstruktur Feld verwendet wurde.

5 Kinoverwaltung (aus Abitur 2020)

Das Kino „Altstadtklotz" benötigt ein Verwaltungssystem und gibt dazu eine Software in Auftrag. [...]

Nach einem Jahr Produktivbetrieb will „Altstadtklotz" seine Software um ein Bewertungssystem erweitern lassen. Kunden sollen zu einem Film, den sie gesehen haben, einen kurzen Bewertungstext schreiben und einen bis fünf Bewertungssterne vergeben können. Es wird vorgeschlagen, zur Verwaltung der Bewertungen in einer gemeinsamen Liste für alle Filme folgendes Klassendiagramm zugrunde zu legen:



Die Entwickler entschieden sich, die Bewertungseinträge nicht direkt, sondern über eine zusätzliche Klasse KNOTEN zu verketten. Nennen Sie das zugrunde liegende Konzept sowie zwei damit verbundene Vorteile.

***6 Grafische Benutzeroberfläche für das Projekt**

- **a** Skizzieren Sie für Ihr Projekt aus Aufgabe 2 eine passende grafische Benutzeroberfläche.
- **b** Erstellen Sie eine Klasse, die die grafische Benutzeroberfläche realisiert.
- **c** Stellen Sie die Verbindung zwischen der grafischen Benutzeroberfläche und dem Projekt her. Testen Sie ausführlich!

7 Forschungsauftrag: Entwurfsmuster Model – View – Controller (MVC)

- **a** Das Entwurfsmuster MVC beschreibt den effizienten Aufbau von Projekten mit einer grafischen Benutzeroberfläche. Informieren Sie sich über den Aufbau des Musters.
- **b** Beschreiben Sie, inwieweit darin das Konzept der Trennung nach Aufgabenbereichen zum Tragen kommt.

→ 6.6

8 Alles klar?

- **a** Beschreiben Sie, welchen Vorteil es allgemein hat, wenn man Klassen so konzipiert, dass sie jeweils nur einen Aufgabenbereich haben.
- **b** Begründen Sie, weshalb dies bei der Warteschlange sinnvoll eingesetzt werden kann.

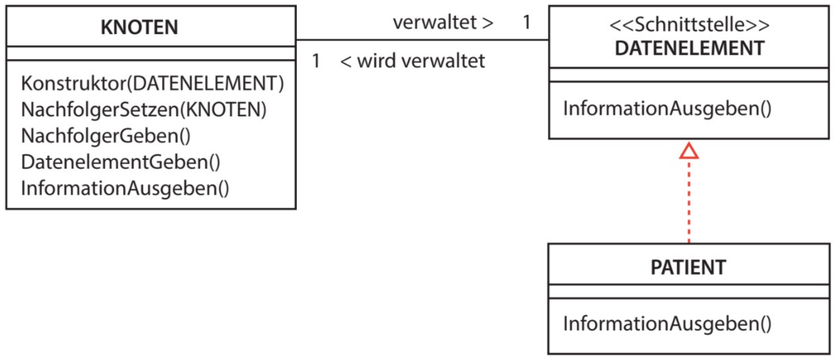
*1.3 Prototyp 2+: Bessere Trennung hilft beim Recycling

In der Vorlage erhalten Sie ein Projekt für eine Warteschlange von Autos in einer Waschstraße. Bauen Sie das Projekt so um, dass es stattdessen die Bürger im Einwohnermeldeamt einer Gemeinde verwalten kann, die auf eine Dienstleistung (z. B. Anmeldung, Ausstellung eines neuen Ausweises...) warten. Zählen Sie, an wie vielen Stellen in den Klassen WARTESCHLANGE und KNOTEN Sie Änderungen durchführen müssen.

Mit der bisherigen Konstruktion der Warteschlange lassen sich nur Objekte einer bestimmten Klasse (z. B. PATIENT) verwalten. Damit die Warteschlange aber in anderen Sachzusammenhängen auch andere Objekte verwalten kann, sollte möglichst eine größere Unabhängigkeit von den verwalteten Daten erreicht werden:

- Die in Warteschlangen verwalteten Datenelemente können ganz unterschiedlich sein: Patienten bei einem Arzt können ebenso verwaltet werden wie Aufträge einer Druckerwarteschlange. Die verwalteten Objekte haben also unterschiedliche Attribute und Methoden – eine Klasse PATIENT z. B. für die Verwaltung spezifischer Patientendaten.
- Daneben haben die verwalteten Objekte Methoden, die sie in ihre Rolle als Teil der Warteschlange beschreiben. In dieser Rolle müssen alle Arten von Datenelementen zumindest eine Methode *InformationAusgeben* zur Verfügung stellen, die zu Kontrollzwecken im Terminalfenster Informationen über das Datenelement ausgibt.

Um diesen Anforderungen gerecht zu werden, muss in Form einer **Schnittstelle** DATENELEMENT ein geeignetes Verbindungsglied bereitgestellt werden, das es Objekten unterschiedlicher Klassen erlaubt, als Teil der Datenstruktur Warteschlange verwendet zu werden. In den Klassen WARTESCHLANGE und KNOTEN treten dann die konkreten Anwendungsklassen nicht mehr als Datentypen auf, sondern nur noch die Schnittstelle.



Die Umsetzung einer solchen Schnittstelle ist abhängig von den Möglichkeiten, die die Programmiersprache bietet. So gibt es beispielsweise in Java ein spezielles Interface-Konzept.

```
interface DATENELEMENT
{
   void InformationAusgeben();
}
```

```
class PATIENT implements DATENELEMENT
{
   ...
   void InformationAusgeben()
   {
        ...
   }
   ...
}
```


In Python dagegen gibt es kein spezielles Interface. Hier kann eine Vererbungsbeziehung zur Umsetzung genutzt werden, man kann aber mittels „pass“ Methoden ohne Rumpf definieren. Eine Umsetzung mittels Vererbung wiederum wäre in Java unmöglich, wenn die Klasse der verwalteten Objekte bereits Unterklasse einer anderen Klasse ist, weil Java im Gegensatz zu Python Mehrfachvererbung unterbindet.

Durch die Verwendung einer geeigneten **Schnittstelle** lässt sich die Datenstruktur Warteschlange in unterschiedlichen Anwendungskontexten ohne Anpassungsarbeiten verwenden.

Aufgaben

1 Schnittstellen

a Informieren Sie sich, wie sich in der von Ihnen verwendeten Programmiersprache Schnittstellen realisieren lassen.

b Stellen Sie die Konzepte Vererbung, Vererbung mit abstrakten Klassen und – falls in Ihrer Programmiersprache vorhanden – Interface gegenüber. Geben Sie jeweils ein Beispiel an und beschreiben Sie sinnvolle Einsatzszenarien.

2 Schnittstellen- und Vererbungsbeziehungen

Eine Schnittstellenbeziehung ist etwas schwächer als eine Vererbungsbeziehung. Beispiele: „Ein Bernhardiner **ist ein** Hund.“ beschreibt eine **Vererbung**sbeziehung. „Ein Patient kann Teil einer Warteschlange sein (es ist aber nicht seine zentrale Aufgabe).“ beschreibt eine Schnittstellenbeziehung. Geben Sie weitere Beispiele für Vererbungs- und Schnittstellenbeziehungen an.

3 Anpassung der Projekte

a Setzen Sie ein Projekt aus dem letzten Kapitel mit Hilfe einer Schnittstelle um. Verwenden Sie diese Schnittstelle auch in den nächsten Kapiteln weiter. Da die Verwendung einer Schnittstelle nicht abiturrelevant ist, wird dies in den nächsten Kapiteln nicht verlangt, auch wenn die Implementierung dadurch eleganter wäre. Sie dürfen dies natürlich weiterhin anwenden, wenn Sie möchten!

b Zeigen Sie durch die Verwendung einer anderen Anwendungsklasse, dass durch die Verwendung der Schnittstelle der Code in den Klassen WARTESCHLANGE und KNOTEN unverändert bleibt.

4 Änderungen im Rollenspiel: Fehlanzeige

Begründen Sie, warum in einem Rollenspiel (siehe S. 22, Aufgabe 1) durch die neue Schnittstelle keine Änderung zu beobachten ist.

5 Forschungsauftrag: Schnittstelle

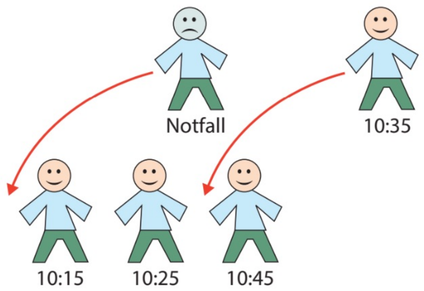
Der Begriff „Schnittstelle“ taucht in der Informatik in unterschiedlicher Bedeutung auf. Entwickeln Sie eine erläuternde Übersicht.

1.4 Prototyp 3: Rekursive Methoden der Liste

Bei dem beigefügten Projekt zur Warteschlange fehlt die Referenz auf das Ende.
a Testen Sie das Programm. Zeigen Sie, dass dennoch Elemente hinten einfügbar sind.
b Analysieren Sie in Zweierteams den Code und finden Sie heraus, wie die Methoden zum Einfügen am Ende hier arbeiten.

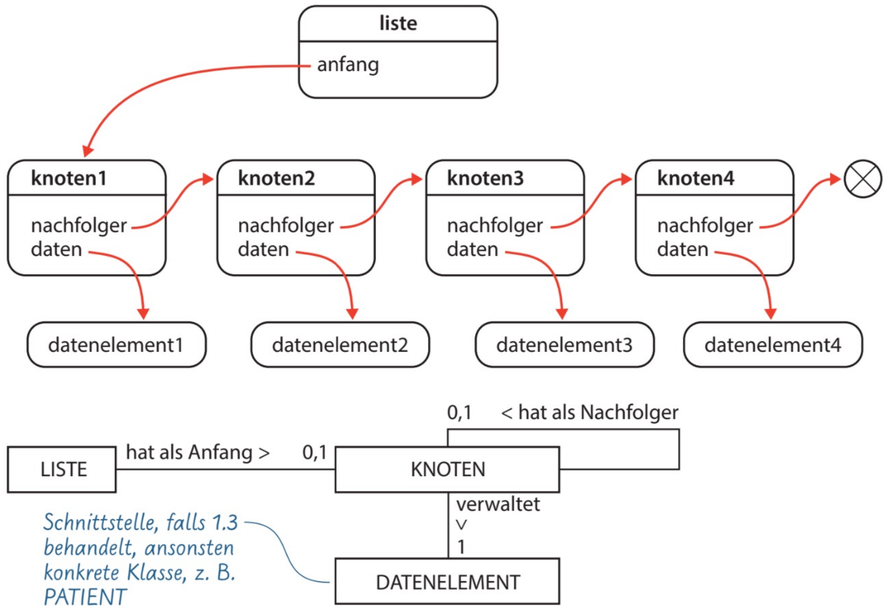
Erweiterte Anforderungen trotz Vereinfachung der Datenstruktur

In der Arztpraxis von Herrn Dr. Sommer weiß man von der gut funktionierenden Warteschlangenverwaltung bei Dr. Ted Tanus. Allerdings hat man zusätzliche Anforderungen:



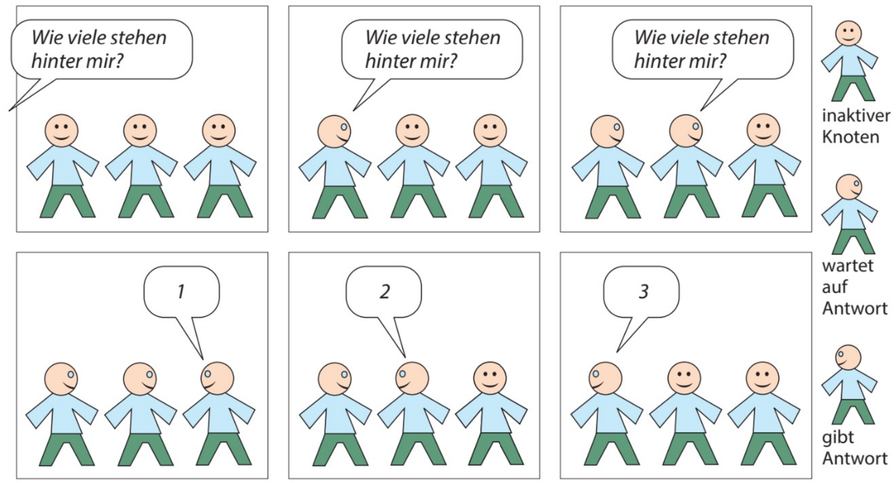
- Die Patienten sollen nach den geplanten Terminen und nicht nach ihrer Ankunftszeit an der passenden Stelle der Warteschlange einsortiert werden, und Notfälle sollen ganz nach vorne rücken können.
- Patienten fragen oft nach: „Wie viele sind noch vor mir?“ Besonders Ungeduldige lassen sich oft einen Ersatztermin geben und verlassen nach einiger Zeit das Wartezimmer wieder.
- Da es die Arzthelfer bei viel Betrieb gelegentlich auch mal versäumen, jemanden einzutragen, soll geprüft werden können, ob ein Patient überhaupt aufgenommen wurde.

Diese Anforderungen gehen über das hinaus, was die Warteschlange bietet. Wieder müssen Objekte in einer bestimmten Reihenfolge verwaltet werden. Neu ist die Forderung, auf alle verwalteten Knoten zugreifen zu können.
Eine Datenstruktur, die die gewünschten Anforderungen erfüllt, heißt **einfach verkettete Liste** (im Folgenden auch kurz: **Liste**).
Wie die Einstiegsaufgabe zeigt, ist es leicht möglich, diesen Anforderungen gerecht zu werden. Und dabei vereinfacht sich sogar noch die Datenstruktur, weil die Referenz auf das Ende nicht mehr notwendig ist und man über die Nachfolger zum Ende gelangen kann. Objekt- und Klassendiagramm modellieren die neue Datenstruktur.



Methoden, die sich selbst aufrufen

Es ist nicht direkt feststellbar, wie viele Datenelemente in der Liste sind, denn das Objekt der Klasse LISTE hat nur auf das erste Element direkten Zugriff. Weder die Liste selbst noch die einzelnen Knoten tragen Information über die Anzahl der Datenelemente (kurz: Länge) der Liste. Ein eigenes Attribut bei der Liste wird auch nicht gewünscht, da die Zahl eine untergeordnete Bedeutung für die Liste hat und die notwendigen Änderungen am Attributwert den Code der betroffenen Methoden unnötig verlängern.
Jeder Knoten hat nur Sicht auf seinen unmittelbaren Nachfolger. Auf die Frage „Wie viele Knoten sind hinter mir?“ kann kein Knoten direkt eine Antwort geben – außer dem letzten Knoten, der weiß, dass er keinen Nachfolger hat. Der Zugriff auf die Knoten über die Liste erfolgt aber über den ersten Knoten. Dieser kann die Frage also – wenn er nicht gleichzeitig der letzte ist – nicht direkt beantworten. Er kann die Frage aber an den Nachfolger weiterleiten und so lange warten, bis von diesem eine Antwort kommt.



Kennt auch der Nachfolger die Antwort nicht, kann dieser nichts anderes tun als weiterzufragen. Dies geht so lange, bis der letzte Knoten erreicht ist. Der letzte Knoten kann erkennen, dass er selbst keinen Nachfolger hat, und liefert den Wert 1 an seinen auf Antwort wartenden Vorgänger zurück. Der Vorletzte erhöht den ihm mitgeteilten Wert um eins auf 2 und meldet „2“ an den Drittletzten zurück. Dieser zählt wieder eins dazu und gibt „3“ als Antwort an den wartenden Vorgänger zurück. Der Vorgang wiederholt sich so lange, bis schließlich der vorderste Knoten die Länge der Liste zurückgibt.
Um bei der Klasse LISTE eine Funktion *LängeGeben* zu erhalten, muss man also zunächst in der Klasse KNOTEN eine Funktion *RestlängeGeben* implementieren. Diese arbeitet nach dem folgenden Schema: Hat das Attribut nachfolger eine leere Referenz, wird der Wert 1 zurückgegeben. Ansonsten wird die Funktion *RestlängeGeben* für den Nachfolger aufgerufen und der Wert, den diese Funktion liefert, wird um eins erhöht.
Die Funktion *LängeGeben* unterscheidet sich von *RestlängeGeben* nur im Referenzattribut und in den Zahlenwerten.

Klasse LISTE
Methode LängeGeben() -> GANZZAHL

```
falls anfang == leer dann
      return 0
sonst
      return anfang.RestlängeGeben()
endefalls
```

Klasse KNOTEN
Methode RestlängeGeben() -> GANZZAHL

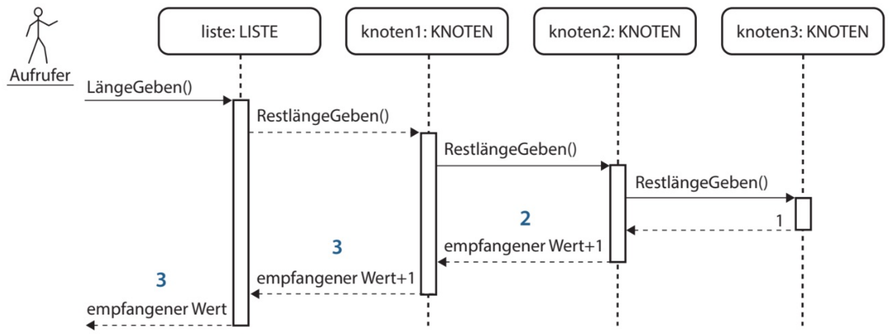
```
falls nachfolger == leer dann
      return 1
sonst
      return nachfolger.RestlängeGeben()+1
endefalls
```

Abbruchbedingung
Rekursionsende (Abbruchbedingung wahr)
rekursiver Aufruf (Abbruchbedingung falsch)

Bemerkenswert an der Funktion *RestlängeGeben* ist, dass im Methodenrumpf wiederum ein Aufruf der Methode *RestlängeGeben* notiert ist – die Methode ruft sich selbst (auf einem anderen Objekt) auf. Diese Vorgehensweise nennt man **rekursiv**.
Das Prinzip einer solchen **Rekursion** besteht darin, dasselbe Verfahren auf eine gleichartige Problemsituation anzuwenden (**Rekursionsschritt**), die sich durch jede Anwendung immer stärker vereinfacht (hier: eine immer kürzere Teilliste). Im Verlauf der Abarbeitung wird eine **Abbruchbedingung** erfüllt (hier: `nachfolger == leer`); das **Rekursionsende** ist erreicht und die Lösung (hier: 1) kann direkt zurückgegeben werden.

Im Beispiel wird die Funktion *RestlängeGeben* für immer kürzere Listenteile aufgerufen, bis schließlich der letzte Knoten (ein Listenteil der Länge 1) erreicht ist. Die Abbildung unten zeigt diese Aufrufsequenz. Auf der Basis der direkt lösbaren Problemsituation am Ende der Liste können dann auch die vorherige Situation und im Anschluss daran alle anderen vorhergehenden Situationen (hier jeweils durch Erhöhung des Rückgabewertes um 1) sukzessive gelöst werden.

Grundlegendes zum Sequenzdiagramm ist im Abschnitt „Startklar" beschrieben.



Bemerkung: Rekursion birgt die Gefahr, dass ein Methodenaufruf niemals endet, wenn nicht auf die schrittweise Vereinfachung der Problemsituation geachtet wird.

Eine Liste ist eine verkettete Folge von Knoten, die Referenzen auf den Nachfolger und das verwaltete Datenelement enthalten.
Rekursive Methoden rufen sich selbst so lange auf (**Rekursionsschritt**), bis eine **Abbruchbedingung** erfüllt ist (**Rekursionsende**). Mit jedem Schritt wird dabei ein Problem auf ein gleiches Problem in einer einfacheren Situation übertragen, bis es schließlich lösbar wird. Durch rekursive Methoden ist es möglich, elegant auf alle Elemente der Liste zuzugreifen und die Liste zu verändern.

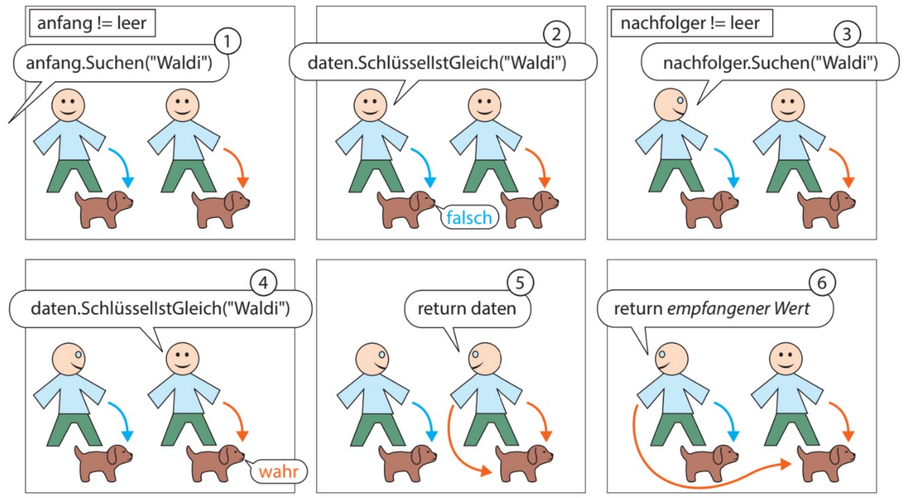
Aufgaben

1 Hinten einfügen
Die Vorlage der Einstiegsaufgabe liefert die Lösung für die Methode *HintenEinfügen*.
- **a** Erproben Sie die Objektkommunikation in einem selbst geplanten Rollenspiel bzw. zeichnen Sie einen Comic wie im Lehrtext.
- **b** Skizzieren Sie ein Sequenzdiagramm für eine Liste, die bereits zwei Elemente enthält.

2 Bestimmung der Länge der Liste
- **a** Erproben Sie die Objektkommunikation zu der Methode aus dem Lehrtext in einem selbst geplanten Rollenspiel.
- **b** Implementieren Sie die Methode im Projekt aus der Einstiegsaufgabe des Kapitels.

3 Überblick gewinnen
- **a** Erstellen Sie ein Klassendiagramm (erweitert auch um Referenzattribute) für die Klassen der Listenstruktur mit allen im Unterricht behandelten Methoden einschließlich Parametern und Rückgabetyp.
- **b** Ergänzen Sie die Methoden zum Suchen passend zum unten abgebildeten Comic.

4 Suchen in der Liste
Der Comic zeigt das Suchen in Pseudocode-Schreibweise. Analysieren Sie ihn und implementieren Sie die Suche in Ihrem Projekt aus der Aufgabe 2 bzw. 3.



Einschub: Programme Testen

Jede Person, die programmiert, weiß es: Der erste Implementierungsversuch ist selten fehlerfrei, geschweige denn perfekt. Schon in den letzten Jahren haben Sie Kompetenzen in diesem Bereich erworben zur Beseitigung von Kompilierzeitfehlern, Laufzeitfehlern und logischen Fehlern, die im Laufe der Programmentwicklung auftreten. Häufig sind die logischen Fehler am schwierigsten zu finden, da bei ihnen keine Fehlermeldung auftritt.
In den letzten Kapiteln haben Sie wahrscheinlich auch viel getestet: Sie haben einzelne Objekte erzeugt, diese in die Liste eingefügt und dann z. B. über die Objektbeziehungen geprüft, ob die Objekte passend über Referenzen verbunden sind und die Methoden korrekt arbeiten. Möglicherweise kam es dabei zu einem Fehler wie hier:

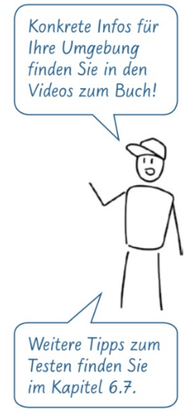
```
l: LISTE
l = neu LISTE()
l.HintenEinfügen(neu PATIENT("Rainer Zufall"))
l.HintenEinfügen(neu PATIENT("Kurt Zarbeit"))
```

Beim Testen einer **Methode mit Rückgabewert** überlegt man sich vorab, wie die Funktion auf die Eingabe bestimmter Werte korrekt antworten sollte. Stimmt dann beim Ablauf des Tests das Ergebnis der Funktion mit der erwarteten Antwort überein, so gilt der Test als bestanden. In der Regel wird nicht nur ein einziger Methodenaufruf ausgeführt, sondern die Methode für verschiedene Eingabewerte, auch z. B. für den Sonderfall einer leeren Liste getestet.

Tests automatisieren
Es ist mühsam und ermüdend, beim Testen von Hand immer wieder mehrere Objekte zu erzeugen sowie Methoden dieser Objekte aufzurufen, bis alle Fehler beseitigt sind. Um hier Zeit und Mühe zu sparen, kann man je nach Sprache Klassen bzw. Skripte schreiben, die den Testvorgang übernehmen. Die Tests können dann nach jeder Änderung durchgeführt werden, um zu prüfen, ob eine Änderung am Code nicht einen neuen Fehler eingefügt hat. Die folgende Abbildung zeigt eine beispielhafte Testklasse für die obigen von Hand durchgeführten Tests.

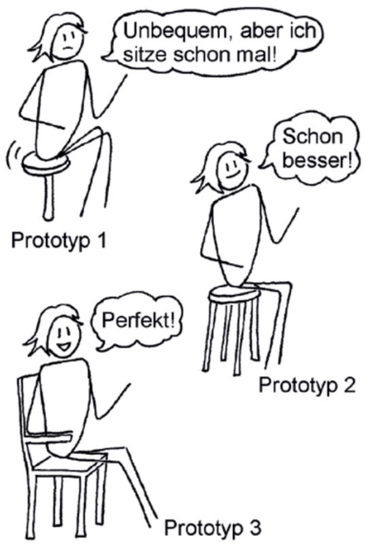
Klasse LISTENTEST
Referenzattribut
`l: LISTE`
Methode TestFürLängeGeben()

`l = neu LISTE()`
`Konsolenausgabe("Test: Kein Patient in Liste:")`
`l.InformationAusgeben()`
`Konsolenausgabe("Test: Länge korrekt?: "+l.LängeGeben()==0)`
`Konsolenausgabe("Test: Ein Patient in Liste:")`
`l.HintenEinfügen(neu PATIENT("Rainer Zufall"))`
`l.InformationAusgeben()`
`Konsolenausgabe("Test: Länge korrekt?: "+l.LängeGeben()==1)`
`l.HintenEinfügen(neu PATIENT("Kurt Zarbeit"))`
`...`



Einschub: Prototyping

In den bisherigen Kapiteln haben wir von der Warteschlange ausgehend in mehreren Schritten eine Liste entwickelt. Jedes Zwischenergebnis war ein getestetes und lauffähiges Produkt, jedoch haben Schritt für Schritt die Funktionalitäten zugenommen. Solche funktionsfähigen Zwischenergebnisse bezeichnet man als Prototypen. In der Softwareentwicklung ist Prototyping ein Prozess, bei dem in mehreren Schritten (Iterationen) ein ausgereiftes Produkt entsteht. Dabei fokussiert man sich in jedem Schritt auf die neu zu erstellende Funktionalität und vermeidet unnötige Details. Dadurch entstehen schnell sichtbare Fortschritte, die (auch vom Kunden) getestet werden können.



Vorteile von Prototyping:
- schnell vorzeigbare Zwischenergebnisse
- Feedback (z. B. durch den Kunden) und Neupriorisierung für den nächsten Prototypen möglich
- Durch Tests fallen Fehler in der Konzeption früh auf und sind leichter zu beseitigen, weil das Programm noch kürzer ist.
- Blick auf das Wesentliche!
- Ein abstraktes Ziel wird frühzeitig konkret!
- Verschiedene Ideen können in kurzer Zeit erprobt werden.

1.5 Endprodukt: Liste mit Entwurfsmuster Kompositum

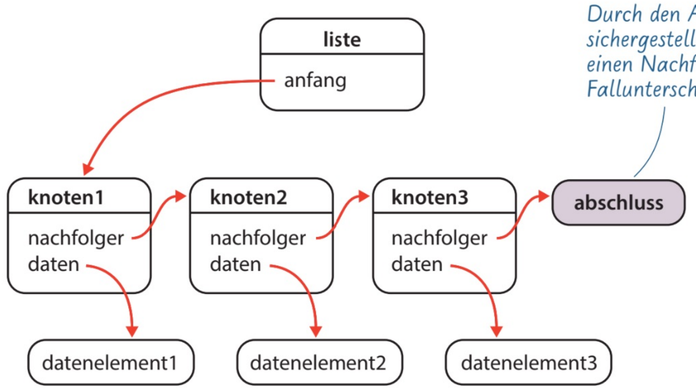
Im beigefügten Projekt wurden alle denkbaren Methoden der Liste umgesetzt.

a Erzeugen Sie ein Objekt der Klasse LISTE und erproben Sie mehrere Methoden, die Sie noch nicht selbst eingesetzt haben.

b Analysieren Sie den Quelltext verschiedener Methoden und formulieren Sie Gemeinsamkeiten der Methoden. Begründen Sie, weshalb es diese Gemeinsamkeiten gibt.

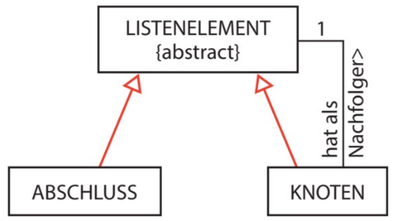
Echt das Letzte!

Bei vielen Methoden aus LISTE und KNOTEN muss man mit bedingten Anweisungen arbeiten: Je nachdem, ob das Ende der Liste erreicht ist oder nicht, reagieren die Methoden unterschiedlich. Um häufige Fallunterscheidungen bezüglich des letzten Elements überflüssig zu machen, bedient man sich eines einfachen, aber wirkungsvollen Tricks: Damit man nicht immer überprüfen muss, ob die Referenz auf den Anfang der Liste oder die Referenz auf den Nachfolger eines Knotens den Wert leer hat, definiert man speziell für das Ende der Liste eine eigene Klasse ABSCHLUSS. Alle anderen Knoten lassen sich weiterhin einheitlich durch die Klasse KNOTEN beschreiben. Der Abschluss verwaltet keine Referenzen, auch nicht auf ein Datenelement.



ABSCHLUSS soll nach außen hin die gleichen Methoden wie KNOTEN zur Verfügung stellen; nur im Inneren der Methoden wird der Unterschied zwischen den beiden Klassen erkennbar: Die Methoden der Klasse ABSCHLUSS haben einzig die Aufgabe, zu regeln, was zu tun ist, wenn das Ende der Liste erreicht ist. Diese Aufgabe fällt bei den Methoden der Klasse KNOTEN nun weg.

Um die gemeinsame Struktur der Methoden herauszustellen und zu verwalten, erhalten KNOTEN und ABSCHLUSS eine gemeinsame abstrakte Oberklasse LISTENELEMENT. Der Abschluss bleibt Bestandteil der Liste, auch wenn die Liste leer ist. In diesem Fall referenziert das Attribut anfang den Abschluss. Da der Nachfolger eines Knotens ein Knoten oder ein Abschluss sein kann, führt die rekursive Beziehung von Knoten zu Listenelement.



Nicht nur bei der Liste, auch bei vielen anderen Problemstellungen, die elementare Objekte zu größeren Strukturen zusammensetzen, ist diese Strukturierung in zwei Klassen mit einer gemeinsamen abstrakten Oberklasse sinnvoll. Zum Umgang mit solchen immer wiederkehrenden Aufgabenstellungen der Softwareentwicklung gibt es eine Reihe von bewährten Standardlösungen,

sogenannte **Entwurfsmuster** (design patterns). Der beschriebenen Aufteilung liegt das Entwurfsmuster **Kompositum** (composite pattern) zugrunde. Die Grundidee des Kompositums ist, in einer abstrakten Klasse primitive Objekte (hier: Abschluss) und Objekte zusammenzufassen, die wieder mit Objekten in Beziehung stehen, die von der abstrakten Klasse abgeleitet sind. Die beiden Arten von Objekten können dadurch einheitlich behandelt werden.

***LängeGeben* viel einfacher!**

Die veränderte Datenstruktur ermöglicht deutlich einfacheren Code bei vielen Methoden der Liste. Bei der Methode *LängeGeben* der Liste wird etwa die bedingte Anweisung überflüssig, weil auch eine leere Liste ein Abschluss-Listenelement hat und von diesem den Wert 0 erhält.
Auch ein Knoten braucht keine bedingte Anweisung mehr – die rekursiv aufgerufene Methode *RestlängeGeben* referenziert am Ende auf den Abschluss, der als Restlänge immer 0 zurückgibt. Der Knoten kann bei der Restlänge immer 1 addieren, auch als letzter Knoten in der Liste.
Die folgende Tabelle zeigt den deutlich einfacheren Aufbau dieser beiden Methoden, der sich aus der Verwendung der neuen Datenstruktur ergibt.

ohne Kompositum

Klasse LISTE
Methode LängeGeben() -> GANZZAHL

```
falls anfang == leer dann
 return 0
sonst
 return anfang.RestlängeGeben()
endefalls
```

Klasse KNOTEN
Methode RestlängeGeben()-> GANZZAHL

```
falls nachfolger == leer dann
 return 1
sonst
 return nachfolger.RestlängeGeben()+1
endefalls
```

mit Kompositum

Klasse LISTE
Methode LängeGeben() -> GANZZAHL

```
return anfang.RestlängeGeben()
```

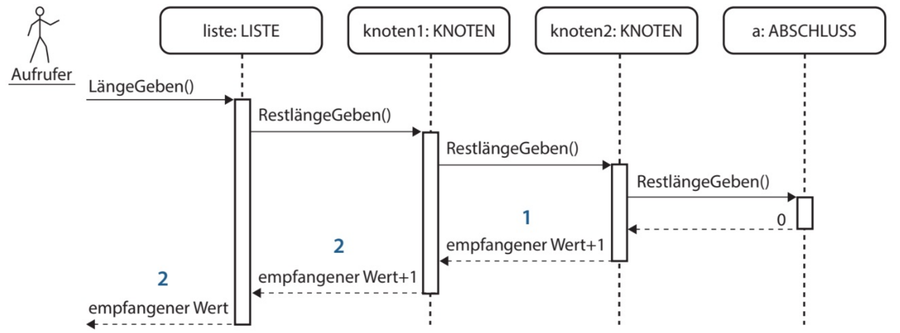
Klasse KNOTEN
Methode RestlängeGeben()-> GANZZAHL

```
return nachfolger.RestlängeGeben()+1
```

Klasse ABSCHLUSS
Methode RestlängeGeben()-> GANZZAHL

```
return 0
```

Die Objektkommunikation mit Kompositum zeigt das folgende Sequenzdiagramm für eine Liste der Länge 2:

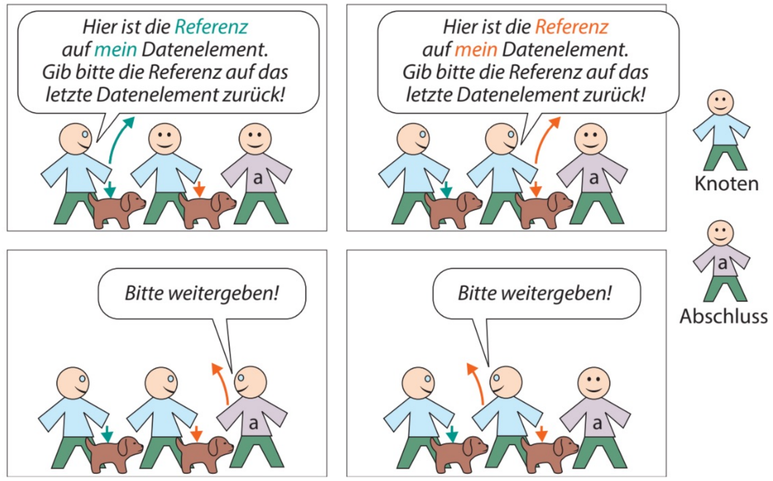


Ein weiteres Beispiel: Entfernen des letzten Knotens

Ein weiteres, komplexeres Beispiel ist das Entfernen des letzten Knotens verbunden mit der Rückgabe des zugehörigen Datenelements. Diese Aufgabe lässt sich in zwei Teile zerlegen:

1. Das Datenelement des letzten Knotens wird ausgehend von der Liste gesucht und an die Liste zurückgegeben.
2. Die Liste lässt den Knoten entfernen, dessen Datenelement es im ersten Schritt empfangen hat und gibt danach das Datenelement zurück.

Beim ersten Schritt gibt es die Schwierigkeit, dass der letzte Knoten nicht weiß, dass er der letzte Knoten ist. Der Abschluss muss diese Aufgabe übernehmen und dafür sorgen, dass die Referenz des letzten Knotens nach vorne durchgereicht wird.



Nach diesen Überlegungen wird der Code selbst aber wieder sehr einfach:

Klasse KNOTEN
Methode EndeGeben(d: DATENELEMENT) -> DATENELEMENT

```
return nachfolger.EndeGeben(daten)
```

Klasse ABSCHLUSS
Methode EndeGeben(d: DATENELEMENT) -> DATENELEMENT

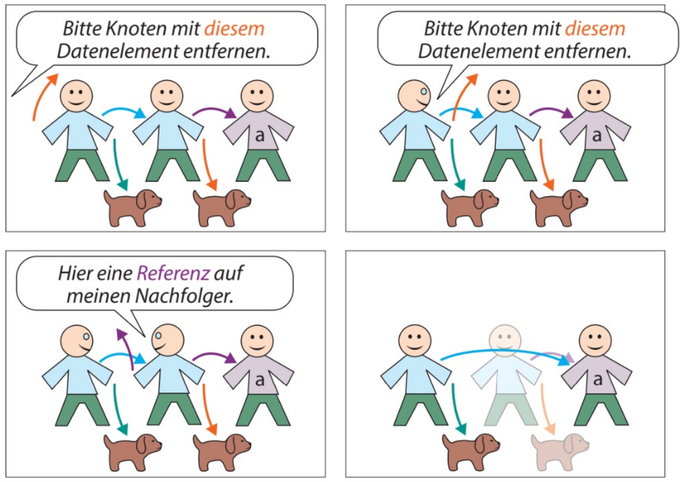
```
return d
```

Der zweite Schritt ist das Entfernen des Knotens, dessen Datenelement die Liste gerade erhalten hat, denn dieses kam vom letzten Knoten. Zurückgeliefert wird jeweils der neue Nachfolger. Jeder Knoten liefert sich selbst zurück, damit er in der Liste bleibt. Nur der letzte Knoten, identifiziert durch sein Datenelement, liefert seinen Nachfolger zurück.

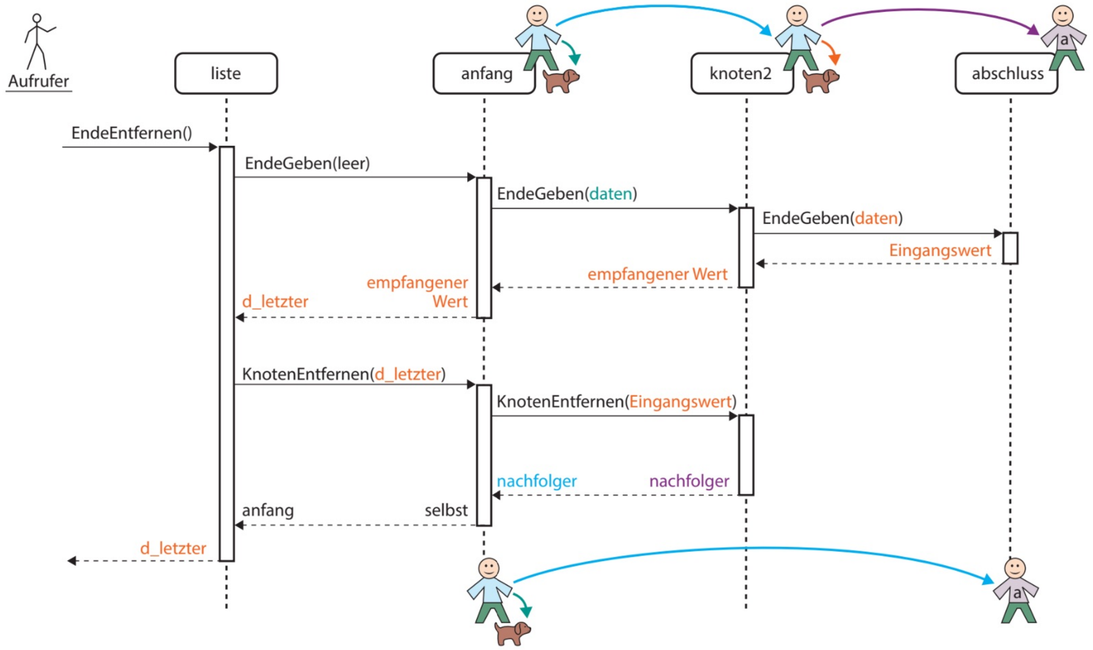
Klasse KNOTEN
Methode KnotenEntfernen(d: DATENELEMENT) -> LISTENELEMENT

```
falls daten == d
dann
 return nachfolger
sonst
 nachfolger = nachfolger.KnotenEntfernen(d)
 return selbst
endefalls
```


Die folgende Abbildung illustriert das Verfahren im Comic:



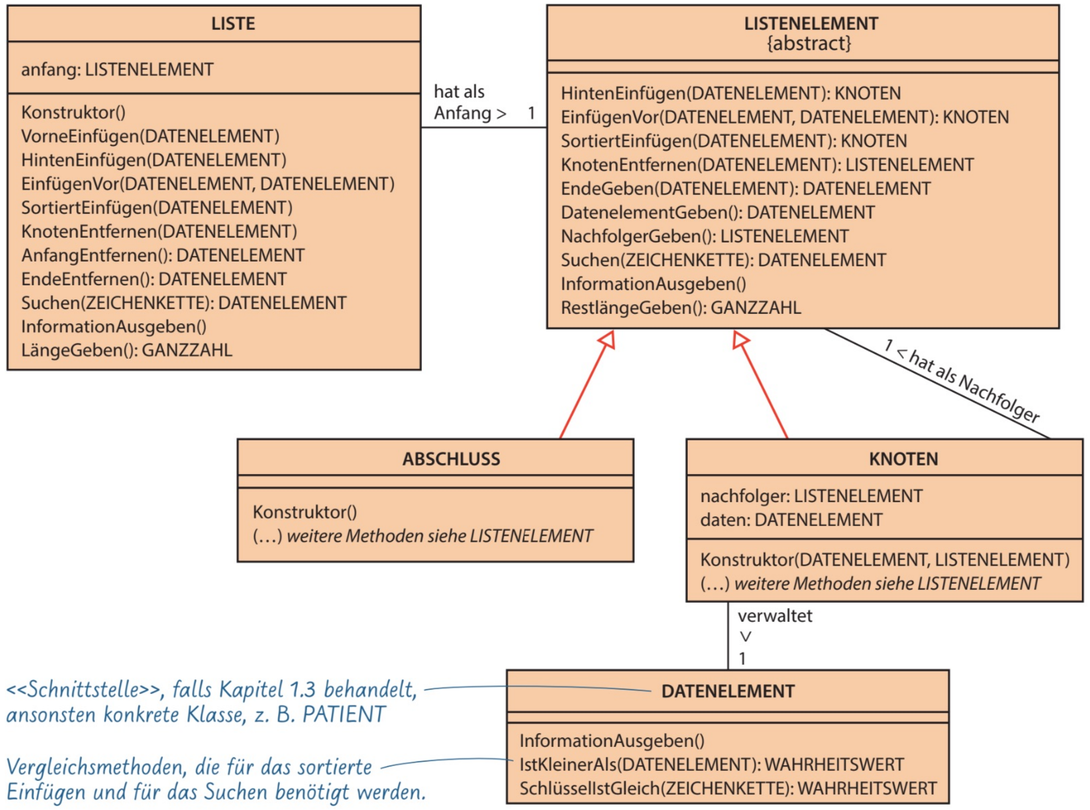
Das folgende Sequenzdiagramm fasst die Objektkommunikation für das Beispiel einer Liste mit zwei Knoten zusammen:



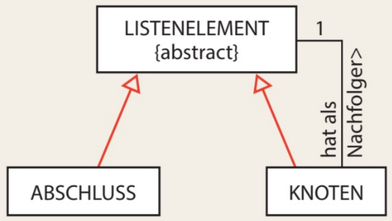
Wenn der letzte Knoten aus einer leeren Liste entfernt werden soll, wird die Methode *KnotenEntfernen* des Abschlusses aufgerufen. In diesem Fall übergibt der Abschluss einfach eine Referenz auf sich selbst und alles bleibt wie gewünscht beim Alten!

Weitere Methoden in der Umsetzung

Das nachfolgende Klassendiagramm, erweitert um Referenzattribute und Datentypen, zeigt die Gesamtstruktur mit weiteren häufig benutzten Methoden der Liste.



Wird das Ende einer Liste von einem Objekt der Klasse ABSCHLUSS markiert, gelingt eine besonders einfache Umsetzung der Methoden.
Bewährte Standardlösungen in der Softwareentwicklung werden **Entwurfsmuster** genannt. Das Entwurfsmuster **Kompositum** hat für die Listenstruktur die nebenstehende Form.



Aufgaben

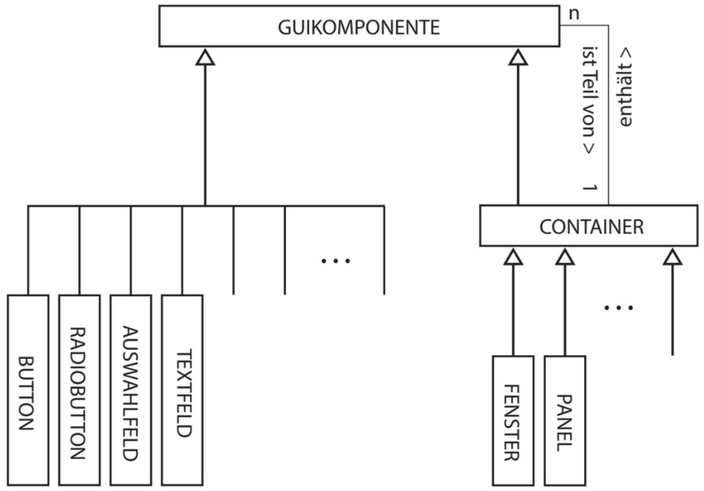
1 Von der Warteschlange zur Liste

Die Prototypen in Kapitel 1.1 – 1.4 waren Vorbereitung für das zentrale abiturrelevante Kapitel 1.5, in dem die Liste voll ausgebaut ist. Stellen Sie die wesentlichen Denkschritte geeignet zusammen (z. B. Cheatsheet oder Mindmap).

2 Kompositum in anderen Anwendungen

Das Entwurfsmuster Kompositum lässt sich auch in anderen Kontexten anwenden.
Zeichnen Sie jeweils ein passendes Klassendiagramm:

- **a** Ein Geschenkartikelhersteller produziert Geschenkartikel und auch Kartons zur Verpackung. Sowohl Geschenkartikel als auch Kartons sind also Produkte.
- **b** Dateisystem mit den Klassen DATEISYSTEMELEMENT, ORDNER, DATEI
- **c** Beschreiben Sie den Sachkontext der nachfolgenden Grafik. Begründen Sie, an welchen Stellen abstrakte Klassen gewählt werden sollten. Erläutern Sie, wo das Entwurfsmuster Kompositum zu finden ist.



Das Verständnis der Liste mit Kompositum ist zentral für die Weiterarbeit im Kapitel 2 und auch für die Vorbereitung auf das Abitur. Es lohnt sich daher besonders, die zentralen Aufgaben 3 – 11 selbstständig (gerne auch im Pair Programming) und konzentriert zu bearbeiten. Nutzen Sie dabei die Chance, offene Fragen mit Mitschülerinnen und Mitschülern und der Lehrkraft zu diskutieren! Wenn Sie schnell sind, suchen Sie nach alternativen Implementierungen.

3 Grundstruktur des Kompositums

- **a** Implementieren Sie die Grundstruktur der Liste nach dem Entwurfsmuster Kompositum mit allen Konstruktoren ohne weitere Methoden. Orientieren Sie sich dabei am Klassendiagramm auf der Vorseite. Die leere Liste erhält schon im Konstruktor einen Abschluss als Anfang der Liste. Wählen Sie ein geeignetes Anwendungsszenario (z. B. Arztpraxis).

b Implementieren Sie die (besonders einfache, weil nichtrekursive) Methode *VorneEinfügen(DATENELEMENT d)*. Diese wird nur in der Klasse LISTE implementiert. Die LISTE erzeugt einen neuen Knoten, der den bisherigen Anfang der Liste als Nachfolger und das übergebene d als Daten erhält. Dieser Knoten wird als neuer Anfang gesetzt.
c Erstellen Sie eine Testklasse mit mindestens drei Knoten und testen Sie damit, ob die Methode *VorneEinfügen* korrekt arbeitet.

4 Bestimmung der Länge der Liste

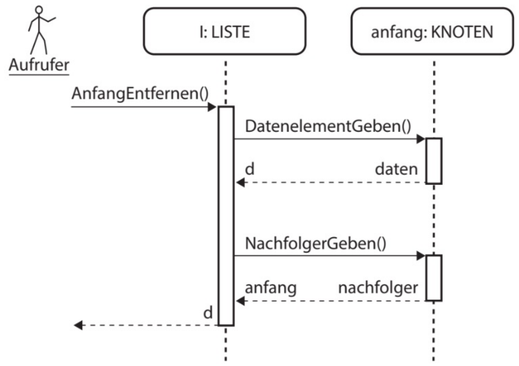
a Erproben Sie die Objektkommunikation zu der im Lehrtext beschriebenen Längenbestimmung in einem selbst geplanten Rollenspiel.
b Implementieren Sie die Methoden im Projekt aus Aufgabe 3. Testen nicht vergessen!
c Skizzieren Sie ein Sequenzdiagramm für eine Liste mit zwei Knoten beim Aufruf von *LängeGeben*.

5 Ende entfernen

a Erproben Sie die Objektkommunikation zu der im Lehrtext beschriebenen Methode *EndeEntfernen* in einem selbst geplanten Rollenspiel.
b Implementieren Sie die Methoden im Projekt aus Aufgabe 3. Testen nicht vergessen!
c Für Schnelle: Formulieren Sie eine alternative Implementierung unter Nutzung von *LängeGeben()*.

6 Anfang entfernen

a Zur Vorbereitung der Methode *AnfangEntfernen() -> DATENELEMENT* benötigen Sie die Standard-Getter-Methoden *DatenelementGeben()* und *NachfolgerGeben()* in der Klasse KNOTEN. Der ABSCHLUSS benötigt diese Methoden auch; es können leer-Werte zurückgegeben werden. In der Oberklasse sind die Methoden abstrakt. Implementieren Sie die Methoden.
b Diskutieren Sie die durch nebenstehendes Sequenzdiagramm beschriebene Objektkommunikation. Begründen Sie, welche der auftretenden Variablen Referenzattribute, welche lokale Variablen sind.
c Implementieren Sie die Methode *AnfangEntfernen* und testen Sie, auch im Fall einer leeren Liste.



7 Information ausgeben

Die Methode *InformationAusgeben()* der Liste soll dafür sorgen, dass auf rekursivem Wege über die Knoten für alle Datenelemente die gleichnamige Methode aufgerufen wird, die auf der Konsole die Attributwerte ausgibt.
a Skizzieren Sie ein Sequenzdiagramm für eine Liste mit zwei Knoten und Datenelementen. Abschluss nicht vergessen!
b Implementieren Sie *InformationAusgeben* in allen beteiligten Klassen. Testen Sie!
c Variieren Sie die Methode so, dass die Information in umgekehrter Reihenfolge der Knoten ausgegeben wird. Tipp: Es genügt, zwei Codezeilen zu vertauschen.

8 Hinten einfügen

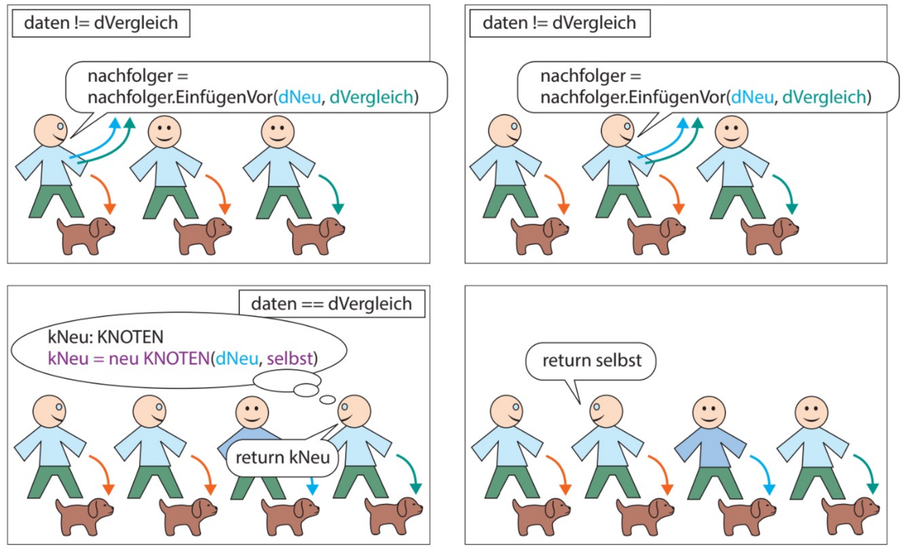
a Am Ende der Liste soll ein neues Datenelement eingefügt werden. Gegeben ist der nachfolgende Pseudocode in der Klasse ABSCHLUSS. Geben Sie an, was der Vorgänger des Abschlusses mit dem Rückgabewert machen muss.

Klasse ABSCHLUSS **Methode HintenEinfügen(d: DATENELEMENT) -> KNOTEN**
`return neu KNOTEN(d, selbst)`

b Die Klasse KNOTEN benötigt den gleichen Methodenkopf wie ABSCHLUSS. Begründen Sie, was der Knoten zurückgeben muss.
c Planen Sie ein Rollenspiel für die Objektkommunikation und führen Sie dieses durch.
d Implementieren Sie die Methode *HintenEinfügen* in allen Klassen der Listenstruktur.

9 Einfügen vor einem bestimmten Datenelement

Der folgende Comic zeigt die Objektinteraktion, wenn ein Datenelement vor einem bestimmten anderen in die Liste aufgenommen werden soll. Der Abschluss ist der Einfachheit halber weggelassen.



a Erarbeiten Sie (z. B. nach dem Verfahren Think – Pair – Share) den Pseudocode der Klasse KNOTEN und legen Sie eine geeignete Strategie fest, wenn die Methode für das Abschlussobjekt aufgerufen wird.
b Implementieren Sie die erforderlichen Methoden in allen Klassen der Listenstruktur.

10 Suche

Die Methode *Suchen(ZEICHENKETTE schlüssel) -> DATENELEMENT* sucht ein Datenelement mit einem bestimmten Schlüsselwert und gibt dieses zurück. Ein Datenelement muss dazu eine passende Vergleichsmethode *SchlüsselIstGleich(ZEICHENKETTE schlüssel) -> WAHRHEITSWERT* bereitstellen. Implementieren Sie die passenden Methoden.

11 Sortiertes Einfügen

Das sortierte Einfügen funktioniert ähnlich wie das Einfügen vor einem bestimmten Datenelement, nur die Vergleichsmethode ist eine andere. Das Datenelement benötigt eine Methode *IstKleinerAls(DATENELEMENT) -> WAHRHEITSWERT,* die den Vergleich zweier Datenelemente ermöglicht.

Für Zahlenwerte ist der Vergleich mit dem üblichen Vergleichsoperator „<“ möglich. Für Zeichenketten stellen die Programmierbibliotheken geeignete Vergleichsmethoden bzw. Operatoren bereit (Beispiele siehe Kasten).

Implementieren Sie die Methoden in der Listenstruktur.

java.lang.String:

```
int compareTo(String)
"Alm".compareTo("Zoo")=-25<0
"Udo".compareTo("Udo")=0
"Zug".compareTo("Zahn")=20>0
```

Python, Swift:
übliche Vergleichsoperatoren

12 Wechsel des Anwendungsbeispiels

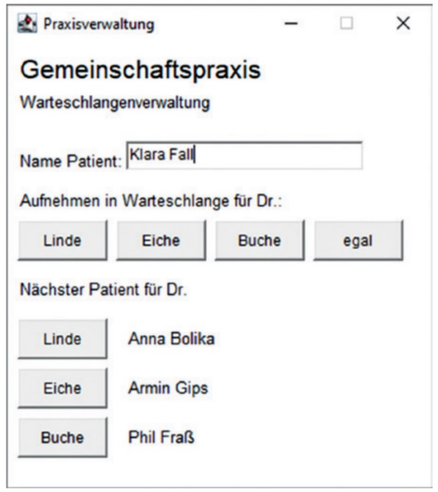
→ 1.3

Erproben Sie die erstellte Liste in einem anderen Anwendungskontext, den Sie bereits genutzt haben (Diashow, Audio- bzw. Videoplayer, Snake...). Vereinfachen Sie den Wechsel ggf. durch Verwendung einer Schnittstelle.

13 Ärztliche Gemeinschaftspraxis

In einer ärztlichen Gemeinschaftspraxis kann man entscheiden, ob man zu einem bestimmten Arzt oder einer bestimmten Ärztin will oder ob es egal ist, von wem man behandelt wird. Beim Aufrufen des nächsten Patienten werden nur diejenigen berücksichtigt, die nicht zu anderen Ärztinnen und Ärzten wollen.

Entwerfen Sie ein Programm unter Verwendung der Grundstruktur (nach Aufgabe 11).



14 Lotto

Im beigefügten Projekt soll eine sortierte Liste mit Lottozahlen erzeugt werden. Korrigieren Sie vorhandene Fehler und diskutieren Sie auch, welche Möglichkeiten zur Vereinfachung des Projekts bestehen und welche Vor- und Nachteile diese haben.

15 Klassenliste

Erstellen Sie auf der Basis der Grundstruktur (nach Aufgabe 11) ein Programm zum Erfassen und Ändern einer Klassenliste. Die Liste soll zuerst nach Nachname, dann nach Vorname sortiert werden.

16 Fußball-Bundesliga

In der Fußball-Bundesliga erhält eine Mannschaft für einen Sieg drei Punkte, für ein Unentschieden einen Punkt. Erstellen Sie ein Programm auf der Basis der Grundstruktur (nach Aufgabe 11), mit dem es möglich ist, den Punktestand jeder Mannschaft zu speichern und die Punkte eines neuen Spieltages hinzuzurechnen. Nach jeder neuen Eingabe soll die Liste sortiert sein.

17 Snake – Level 2 (Projektaufgabe)

Im Level 2 des Spiels Snake laufen Mäuse über den Bildschirm und werden von der Schlange als neue Glieder aufgenommen, wenn sie an beliebiger Stelle auf das Tier treffen. Daneben fahren auch Autos über das Spielfeld und können den hinteren Teil der Schlange entfernen; der vordere Teil bewegt sich weiter.

Entwerfen Sie in Gruppen ein Modell auf der Basis von Level 1 (siehe S. 17, Aufgabe 5) und setzen Sie dieses um. Konkretisieren Sie dabei die Aufgabenstellung, wo dies notwendig erscheint.

18 Nachträgliches Sortieren durch Einfügen

Eine unsortierte Liste soll nachträglich sortiert werden. Der rekursive Algorithmus *NachträglichSortieren* arbeitet nach dem folgenden Prinzip:

- Eine Liste der Länge 1 ist immer sortiert.
- Von einer längeren Liste wird das vordere Datenelement entfernt. Für die verbleibende Liste wird *NachträglichSortieren* aufgerufen. Ist dies geschehen, wird das zuvor entfernte Element sortiert eingefügt.

Setzen Sie den beschriebenen Algorithmus um und testen Sie!

19 Forschungsauftrag: Liste – iterativ!

a Der folgende Pseudocode zeigt eine iterative Formulierung der Methode *LängeGeben.* Implementieren Sie die Methode in Ihrer Listen-Grundstruktur. Vergleichen Sie die Methode mit der rekursiven Methode *LängeGeben* und erörtern Sie Vor- und Nachteile beider Implementierungen.

Klasse LISTE
Methode LängeGeben2() -> GANZZAHL

```
i: GANZZAHL
i = 0
nächster: KNOTEN
nächster = anfang
solange nicht (nächster istein Abschluss)
  i = i + 1
  nächster = nächster.NachfolgerGeben()
endesolange
return i
```

b Erstellen Sie bei weiteren Methoden eine alternative iterative Lösung.

20 Forschungsauftrag: Weitere Datenstrukturen

Neben den bereits bekannten bzw. demnächst behandelten Datenstrukturen Feld, Warteschlange, Liste und Baum gibt es zahlreiche weitere. Informieren Sie sich über Hash Maps und Dictionaries (Struktur, Einsatzgebiete, Stärken/Schwächen, ...) und stellen Sie die Erkenntnisse zusammen (z. B. Cheatsheet, Präsentation, ...).

1.6 Stapel und Wartenschlange: Spezialformen der Liste

Im beigefügten Projekt finden Sie eine Implementierung der Warteschlange von Grund auf sowie eine Implementierung unter Nutzung der Liste.

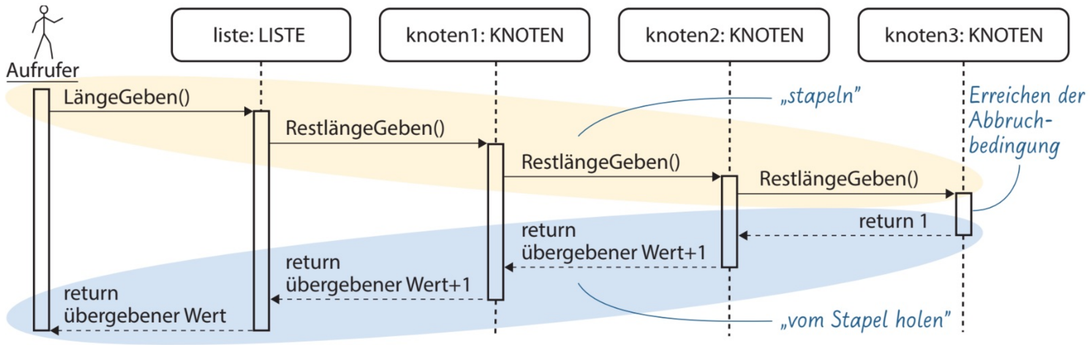
a Korrigieren und ergänzen Sie die Implementierung unter Nutzung der Liste, so dass sie korrekt arbeitet.

b Testen Sie die beiden Varianten und überzeugen Sie sich, dass sie identisch arbeiten.

c Vergleichen Sie den Code und diskutieren Sie Vor- und Nachteile beider Implementierungen.

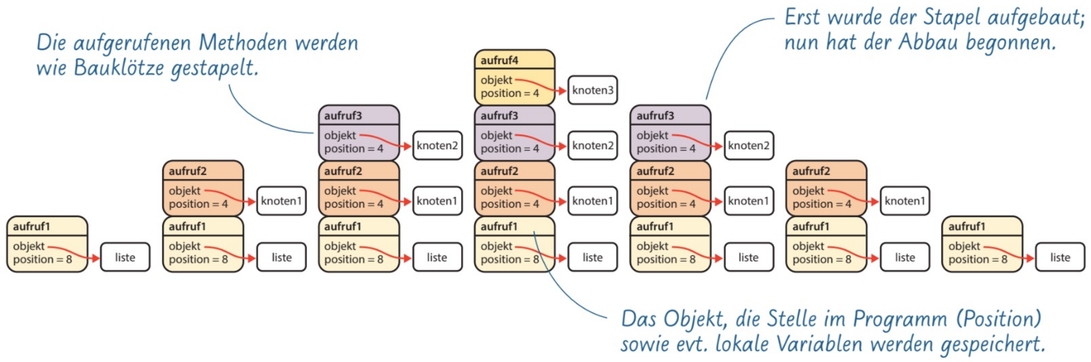
Methoden lassen sich stapeln

Rekursive Methodenaufrufe wurden als eleganter Weg bei der Arbeit mit Listen vorgestellt.



In der Abbildung, die eine Aufrufsequenz aus Kapitel 1.4 beschreibt, ruft die Liste die Methode *RestlängeGeben* für knoten1 auf. Dann muss sie auf Antwort von knoten1 warten, um nach Erhalt dieser Antwort die Methode *LängeGeben* zu beenden. Ebenso ergeht es knoten1, wenn er auf die Rückmeldung seines Nachfolgers wartet.

Man kann sich dies so vorstellen: Nacheinander werden die aufgerufenen Methoden mit allen zugehörigen Informationen oben auf einen **Stapel** gelegt (Abbildung unten). Eine aufrufende Methode verharrt so lange inaktiv auf dem Stapel, bis die darüberliegende aufgerufene Methode ihre Arbeit beendet hat. Dann ist die aufrufende Methode selbst wieder ganz oben auf dem Stapel und kann ihre Ausführung fortsetzen.



Damit diese Fortsetzung korrekt funktionieren kann, muss gespeichert sein, welches Objekt die Methode ausführt und an welcher Stelle (d. h. in welcher Zeile) der Methode die Ausführung unterbrochen wird. Nutzt die Methode lokale Variablen, so muss auch deren Zustand festgehalten werden. Erst wenn die ursprünglich als erstes aufgerufene Methode vollständig ausgeführt ist, verschwindet auch diese vom Stapel.

Diese stapelartige Abarbeitung lässt sich bei allen geschachtelten Methodenaufrufen beobachten, nicht nur bei rekursiven: Erst muss die aufgerufene Methode beendet werden, dann kann die aufrufende Methode ihre Tätigkeit fortsetzen.

→L1

Weitere Anwendungen von Stapelstrukturen

Stapel werden immer dann benötigt, wenn die Ausgabe von Objekten in umgekehrter Reihenfolge zur Aufnahme der Objekte erfolgt (→LIFO-Prinzip). Typische Beispiele sind:

→LIFO: last in, first out

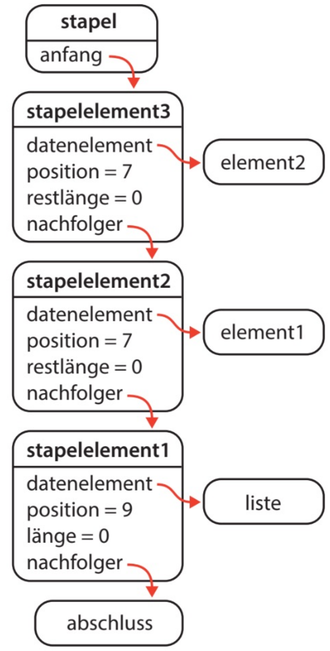
- Verwaltung eines Lagers, bei dem Artikel auch real übereinandergestapelt werden.
- Organisation eines Rangierbahnhofs, bei dem Waggons auf Abstellgleisen geeignet sortiert werden müssen
- Stapel werden auch benötigt, wenn ein Term mit Klammerausdrücken ausgewertet oder auf Korrektheit geprüft werden soll. Zum Weiterlesen 1 beschreibt dies an einem Beispiel.

→L1

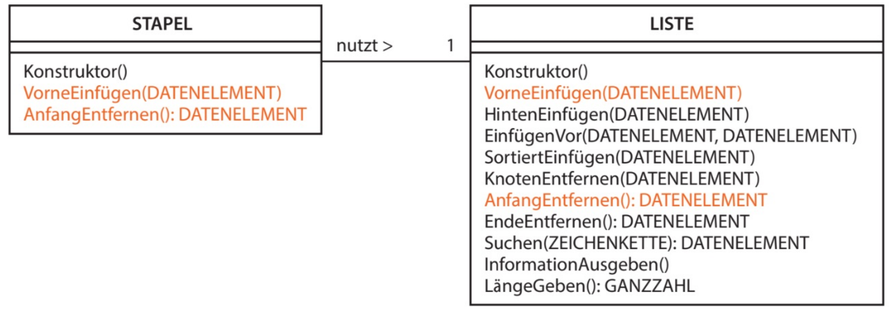
Umsetzung des Stapels

Die Umsetzung eines Stapels in eine geeignete Datenstruktur ist nicht wesentlich komplizierter als im betrachteten Modell: Man benötigt eine Referenz, die auf das oberste Element des Stapels zeigt. Jedes Stapelelement referenziert das darunter liegende. Zum Zugriff auf den Stapel sind nur einfache Methoden erforderlich, die es erlauben, ein Element oben auf den Stapel zu legen und dieses wieder vom Stapel zu entfernen.

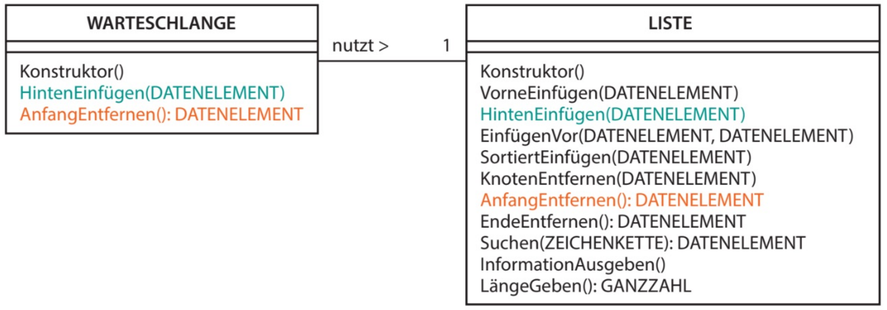
Ein Stapel könnte ähnlich realisiert werden wie die Warteschlange in den ersten Kapiteln. Der einzige Unterschied wäre, dass das Einfügen nun am Anfang und nicht am Ende benötigt würde und dass man deshalb keine Referenz auf das Ende braucht. Der Aufwand für die Umsetzung bliebe aber im Wesentlichen der gleiche.



Eine wesentliche Arbeitsersparnis ergibt sich dagegen bei einem Rückgriff auf die Liste. Vergleicht man die benötigten Methoden von Stapel und Liste, so erkennt man, dass die Methoden des Stapels eine echte Teilmenge der Methoden der Liste sind. Der Stapel ist somit eine Spezialform der Liste. Dies lässt sich am einfachsten realisieren, indem der Stapel ein Referenzattribut vom Typ LISTE nutzt und in den benötigten Methoden die entsprechenden Methoden des Listenobjekts aufruft.



In analoger Weise kann bei vorhandener Implementierung der Liste die Warteschlange erstellt werden, die die Methoden *HintenEinfügen* und *AnfangEntfernen* bereitstellt.



Die Datenstruktur **Stapel** kann Objekte aufnehmen und in entgegengesetzter Reihenfolge zur Aufnahme ausgeben.
Unter Nutzung der Datenstruktur Liste können Warteschlange und Stapel sehr einfach implementiert werden.

Aufgaben

1 Methoden lassen sich stapeln
Die folgende Methode wird für ein Objekt fakultätsberechnung mit dem Wert 4 aufgerufen. Entwickeln Sie ein Stapelmodell wie in der unteren Abbildung auf Seite 42.

```
1  methode Fak(x: GANZZAHL): GANZZAHL
2   falls x == 1 dann
3      return 1
4   sonst
5      return Fak(x-1)*x
6   endefalls
7  endemethode
```

→ L1

2 Termauswertung
Lesen Sie Zum Weiterlesen 1.
Zur Berechnung einer Zufallszahl zwischen 1 und 6 dient die Formel:
SUMME(GANZZAHL(PRODUKT(ZUFALLSZAHL();6));1)
Skizzieren Sie den Verlauf von Operanden- und Operatorenstapel unter der Annahme, dass ZUFALLSZAHL() den Wert 0,3 liefert.

3 Arztpraxis mit elektronischer Patientenakte
Bisher wurde von den Patienten vereinfachend nur der Name gespeichert. In der elektronischen Patientenakte sollen nun außerdem alle Behandlungsdaten gespeichert werden.

a Begründen Sie, weshalb das Speichern der Behandlungsdaten in einem Stapel aus Sicht des Behandelnden besonders geeignet ist, um einen schnellen Überblick über den Patienten zu gewinnen.

b Erweitern Sie das Projekt so, dass der Patient über eine elektronische Patientenakte verfügt, die auch komplett (z. B. auf der Konsole) ausgegeben werden kann.

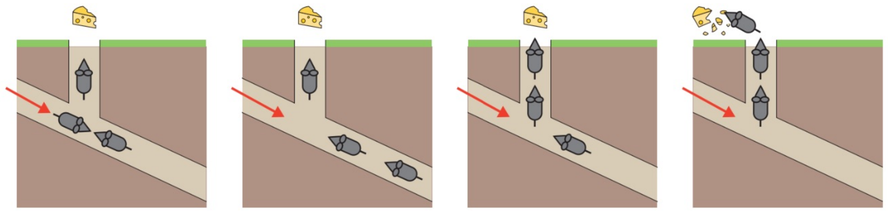
4 Güterbahnhof
Auf einem Rangierbahnhof befinden sich drei Abstellgleise, auf denen Züge aus Waggons zusammengestellt werden können. Wichtigste Information zu jedem Waggon ist sein Zielbahnhof.

a Begründen Sie, weshalb ein Stapel ein geeignetes Modell für ein Abstellgleis ist.

b Entwickeln Sie ein Programm, das es erlaubt, neue Waggons zu erzeugen, Waggons wahlweise auf eines der drei Abstellgleise zu schieben und Waggons von dort zu entfernen.

c (Projektmöglichkeit) Gestalten Sie eine geeignete Benutzerumgebung für das Programm. Überlegen Sie sich selbst sinnvolle Erweiterungen und setzen Sie diese um.

5 Cheese-Champions (aus 17. Bundeswettbewerb Informatik)
Prof. Grips will die Durchsetzungsfähigkeit einzelner Mäuse einer Mäusekolonie feststellen. Dazu hat er die folgende Versuchsanordnung entworfen: Ausgehend von einem unterirdischen Gang gräbt er in einem spitzen Winkel einen Ausgang zur Oberfläche, an dessen Ende er ein Stück duftenden Käse deponiert. Die Mäuse der Kolonie kommen direkt hintereinander von links in den unterirdischen Gang und wollen durch den Abzweig zum Käse. Da der Gang zum Käse in einem spitzen Winkel abzweigt, muss jede Maus erst über den Abzweig hinaus weiter in den Gang hineinlaufen. Dann dreht sie sich um und hat möglicherweise ein Problem: Sie stößt auf die nachfolgende Maus, die ebenfalls in den rechten Abschnitt möchte.



Präziser gelten die folgenden Regeln für das Erreichen des Käses:

- Jede Maus muss zuerst an der Abzweigung vorbei.
- Wenn sie im rechten Teil nach der Abzweigung ist, kann sie entweder den Zweikampf gegen die nachfolgende Maus gewinnen und so in die Abzweigung gelangen oder sie wird weiter in den unterirdischen Gang geschoben. Die nachfolgende Maus wird dann in einen Zweikampf mit der ihr nachfolgenden Maus verwickelt.

Prof. Grips nummeriert die Mäuse in der Reihenfolge, in der sie in den Gang hineinlaufen, mit 1, 2, 3, ... und beobachtet, in welcher Reihenfolge sie beim Käse ankommen.

a Schreiben Sie ein Programm, das aus der beobachteten Reihenfolge, in der die Mäuse beim Käse ankommen, die Anzahl der gewonnenen Zweikämpfe für jede Maus berechnet und ausgibt.

b Zusätzlich soll Ihr Programm nicht realisierbare Austrittsreihenfolgen erkennen und zurückweisen (wie z. B. die Austrittsreihenfolge 3, 1, 2).

Teste dich selbst

T1 Richtig oder falsch?
Beurteilen Sie, ob folgende Aussagen richtig oder falsch sind. Begründen Sie Ihre Meinung bei falschen Aussagen und geben Sie eine berichtigte Aussage an:

a Durch eine Liste lässt sich eine 1:n-Beziehung realisieren.
b Eine Datenstruktur ist immer dann rekursiv, wenn sie rekursive Methoden besitzt.
c Die Trennung von Struktur und Daten geschieht durch Aufteilung der Klasse KNOTEN in KNOTEN und ABSCHLUSS.
d Im Entwurfsmuster Kompositum besitzt die abstrakte Oberklasse LISTENELEMENT die gleichen Methoden wie KNOTEN und ABSCHLUSS: Alle diese Methoden sind in der Oberklasse abstrakt.
e Das Stapelprinzip an sich ist unnötig, weil es ungerecht ist. Wenn man sich irgendwo als Erster anstellt, will man ja auch nicht als Letzter bedient werden.

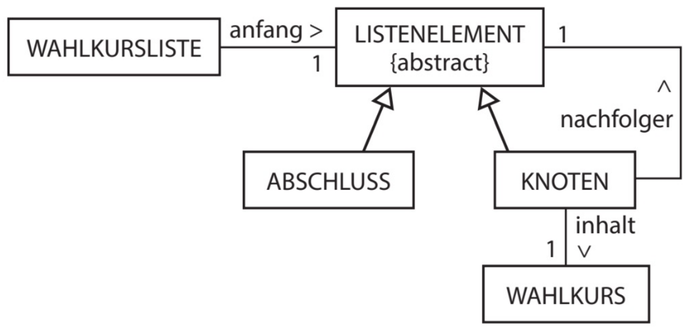
T2 Ich check 's, dank deiner Hilfe!
Ferdi hat im Unterricht nicht gut aufgepasst und braucht dringend Ihre Nachhilfe vor der alles entscheidenden letzten Prüfung. Helfen Sie ihm, indem Sie ihm die folgenden Begriffe erklären und zeigen, wie man damit umgeht: Liste, Warteschlange, Trennung von Struktur und Daten, rekursive Methode, Entwurfsmuster Kompositum, Stapel.

T3 Schulverwaltungssoftware (aus Abitur 2022; Fortsetzung von S. 10)
Die Schule bietet verschiedene Wahlkurse an, die als Objekte der folgenden Klasse WAHLKURS verwaltet werden:

WAHLKURS
name teilnehmerzahl lehrkraft fachgebiet
fachgebietIstGleich(fachgebiet)

Die Methode *fachgebietIstGleich(fachgebiet)* gibt *wahr* zurück, wenn das als Parameter übergebene Fachgebiet mit dem Wert des Attributs *fachgebiet* übereinstimmt, ansonsten *falsch*.

Die Wahlkurse werden in einer einfach verketteten Liste gemäß folgendem Klassendiagramm verwaltet:



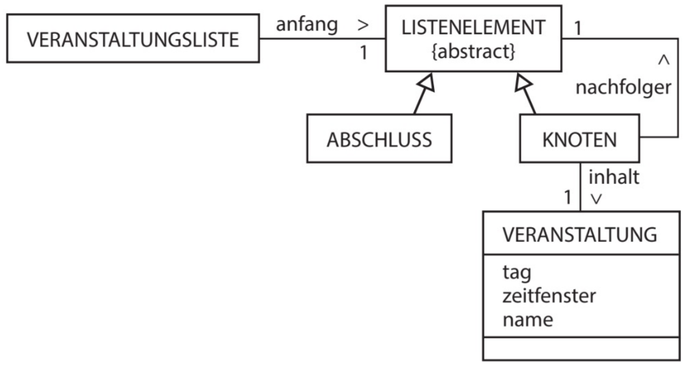
a […] Erläutern Sie […], warum es sinnvoll ist, dabei das Entwurfsmuster Kompositum zu verwenden, und stellen Sie dar, inwiefern bei der gegebenen Umsetzung die Trennung von Struktur und Daten berücksichtigt wurde.
b In der Klasse WAHLKURSLISTE sollen folgende Methoden zur Verfügung stehen:
- *anzahlKurseGeben(fachgebiet)* gibt die Anzahl der Wahlkurse des übergebenen Fachgebiets zurück.

- *zuKleineKurseLoeschen(minTeilnehmerzahl)* entfernt sämtliche Wahlkurs-Objekte aus der Liste, deren Teilnehmerzahl unter der übergebenen Mindestteilnehmerzahl liegt.

Notieren Sie […] eine Implementierung aller Klassen der Listenstruktur sowie eine mögliche Implementierung der beiden oben aufgeführten Methoden und aller dazu nötigen Methoden in der Listenstruktur (ohne Konstruktoren). Verwenden Sie so weit wie möglich das Prinzip der Rekursion.
Dabei können Sie davon ausgehen, dass die oben gegebene Klasse WAHLKURS einschließlich der Methoden zum Geben und Setzen der Attributwerte bereits vollständig implementiert ist.

c Die Schulverwaltungssoftware ermöglicht es, dass verschiedene Personen Wahlkurslisten gemäß der gegebenen Listenstruktur erstellen, die anschließend in eine Gesamtliste zusammengeführt werden. Dazu soll eine rekursive Methode *hintenEinfuegen(wahlkursliste)* zur Verfügung stehen, welche die in der übergebenen Liste *wahlkursliste* enthaltenen Wahlkurse hinten an jene Liste anhängt, von der die Methode ausgeführt wird.
Stellen Sie für eine mögliche Implementierung den Ablauf des Methodenaufrufs *wkliste1.hintenEinfuegen(wkliste2)* in einem Sequenzdiagramm dar, wobei *wkliste1* zwei Knoten *knoten1* und *knoten2* verwaltet. Sie können davon ausgehen, dass Methoden zum Geben und Setzen von Attributwerten zur Verfügung stehen.

T4 Burning Man (aus Abitur 2021; Fortsetzung von S. 11)
Um die Veranstaltungen des Festivals zeitlich gut aufeinander abstimmen zu können, sind pro Tag vier feste, sich nicht überlappende Zeitfenster vorgegeben, in denen Veranstaltungen stattfinden. Die Tage des Festivals sind von 1 bis 9 durchnummeriert, die Zeitfenster von 1 bis 4. Besucher des Festivals können Veranstaltungen online buchen. Die von einem Besucher gebuchten Veranstaltungen werden in einer sortierten Liste gespeichert, wobei die Sortierung zunächst aufsteigend nach Tag und dann aufsteigend nach Zeitfenster erfolgt. Es ist nicht ausgeschlossen, dass mehrere Veranstaltungen gebucht werden, die gleichzeitig stattfinden. Die Liste basiert auf folgendem Klassendiagramm:

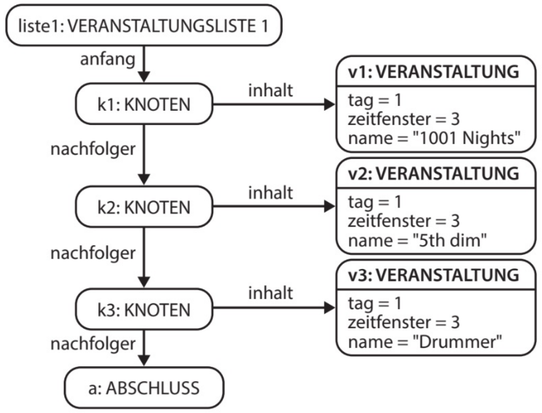


Die Klasse VERANSTALTUNG kann einschließlich der Standardmethoden zum Lesen und Setzen von Attributwerten als implementiert vorausgesetzt werden.

a Die Klasse VERANSTALTUNGSLISTE soll folgende Methoden besitzen:
- *sortiertEinfuegen(veranstaltungNeu)*: fügt eine Veranstaltung in die Liste gemäß obiger Sortierung ein,
- *anzahlGeben(tag, zeitfenster)*: gibt die Anzahl der Veranstaltungen zurück, die der Besucher an diesem Tag im angegebenen Zeitfenster gebucht hat, um ihn so gegebenenfalls darüber zu informieren, dass er Veranstaltungen besuchen will, die gleichzeitig stattfinden.

Notieren Sie […] eine Implementierung der beiden angegebenen sowie der dafür nötigen Methoden aller Klassen der Listenstruktur. Wenden Sie dabei jeweils das Prinzip der Rekursion an.

b Folgendes Objektdiagramm zeigt die Veranstaltungsliste eines Besuchers, der mehrere Veranstaltungen gebucht hat, die gleichzeitig stattfinden.



Gegeben sind folgende Methoden:

In der Klasse VERANSTALTUNGSLISTE:

```
methode m()
  anfang = anfang.m(0, 0)
endemethode
```

In der abstrakten Klasse LISTENELEMENT:

```
abstrakte Methode m(tag, zeitfenster)
```

In der Klasse KNOTEN:

```
methode m(tag, zeitfenster)
  meinTag = inhalt.tagGeben()
  meinZeitfenster = inhalt.zeitfensterGeben()
  nachfolger = nachfolger.m(meinTag, meinZeitfenster)
  falls tag ist gleich meinTag und zeitfenster ist gleich meinZeitfenster
  dann
     gib nachfolger zurück
  sonst
     gib Referenz auf ausführendes Objekt zurück
  endefalls
endemethode
```

In der Klasse ABSCHLUSS:

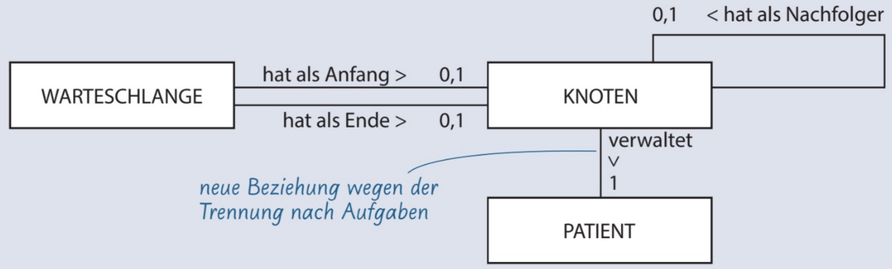
```
methode m(tag, zeitfenster)
  gib Referenz auf ausführendes Objekt zurück
endemethode
```

Stellen Sie in einem Sequenzdiagramm den Ablauf dar, der durch einen Aufruf der Methode m für das Objekt liste1 ausgelöst wird. Beschränken Sie sich auf die Objekte der Klassen VERANSTALTUNGSLISTE, KNOTEN und ABSCHLUSS. Beschreiben Sie zudem, welche Funktionalität die Methode m allgemein hat.

Zusammenfassung

Rekursive Beziehung bei der Warteschlange mit Trennung von Struktur und Daten

Eine Beziehung zwischen Objekten der gleichen Klasse nennt man **rekursive Beziehung**. Zwischen den Elementen, die in der Warteschlange verwaltet werden, besteht die rekursive Beziehung „hat als Nachfolger". Über ein Referenzattribut nachfolger werden die Objekte miteinander verkettet.



Zur besseren **Wartbarkeit** und **Wiederverwendbarkeit trennt** man zwischen **Struktur** (verwaltet in der Klasse KNOTEN mit Referenzen auf Daten und Nachfolger) und **Daten**. Durch Verwendung einer Schnittstelle <<DATENELEMENT>> lässt sich die Wiederverwendbarkeit noch weiter erhöhen.

Rekursive Methoden der Liste – Entwurfsmuster Kompositum

Rekursive Methoden rufen sich selbst so lange auf, bis eine **Abbruchbedingung** erreicht ist, so dass die Selbstaufrufe zu einem Ende kommen. Die rekursive Datenstruktur der Liste ermöglicht den Zugriff auf die Elemente durch rekursive Methoden. Bei Verwendung des **Entwurfsmusters Kompositum** entfallen ansonsten notwendige bedingte Anweisungen zur Überprüfung, ob das Ende der Liste erreicht ist.

Mit Kompositum

Klasse LISTE
Methode LängeGeben() -> GANZZAHL

```
return anfang.RestlängeGeben()
```

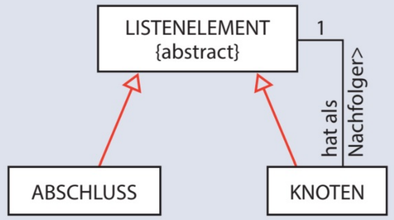
Klasse KNOTEN beerbt LISTENELEMENT
Methode RestlängeGeben() -> GANZZAHL

```
return nachfolger.RestlängeGeben()+1
```

rekursiver Methodenaufruf

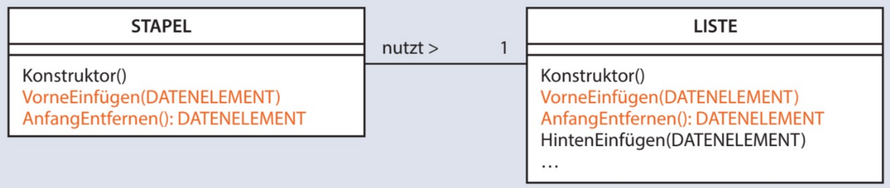
Klasse ABSCHLUSS beerbt LISTENELEMENT
Methode RestlängeGeben() -> GANZZAHL

```
return 0
```



Warteschlange und Stapel als Spezialform der Liste

Warteschlange (**LIFO**-Speicher) und **Stapel** (**FIFO**-Speicher) können unter Nutzung der Liste einfach implementiert werden.



Zum Weiterlesen

L1 Termauswertung

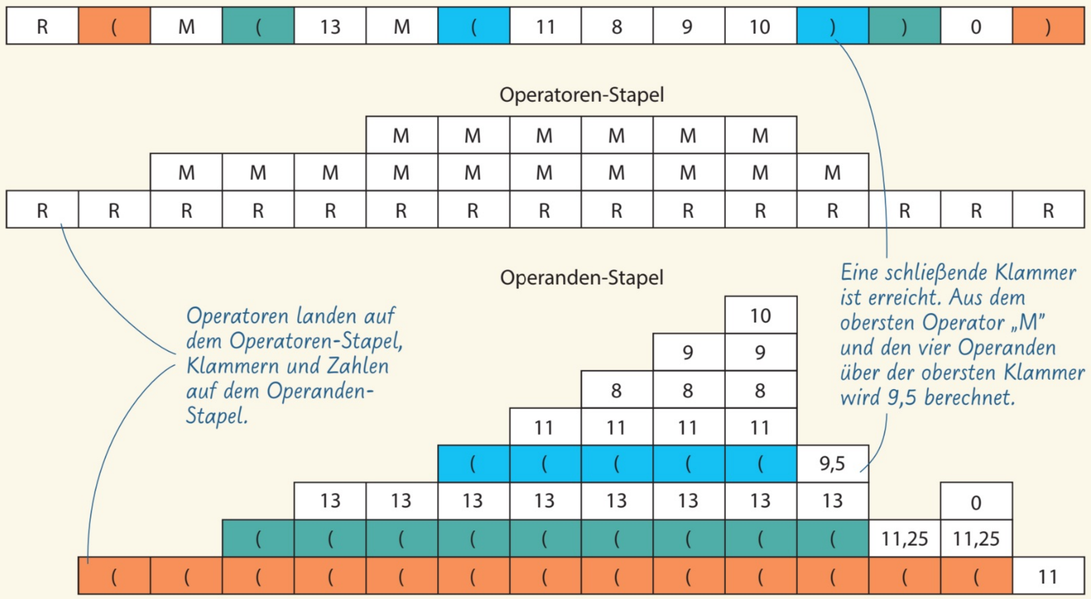
	A	B	C	D
1	**Schulaufgabe**			
2	13			
3	**kleine Leistungsnachweise**			
4	11	8	9	10
5	**Note**			
6	11			

Zur Berechnung einer Halbjahresleistung mit einem Tabellenkalkulationsprogramm wird der Durchschnittswert aus der Punktzahl der Schulaufgabe sowie aus dem Durchschnitt der Punktzahlen der kleinen Leistungsnachweise ermittelt. Das Ergebnis wird gerundet.

Unter Verwendung der konkreten Werte aus der obigen Abbildung und der vordefinierten Funktionen aus dem Tabellenkalkulationsprogramm kann man diesen Mittelwert mit dem folgenden Term berechen:

RUNDEN(MITTELWERT(13;MITTELWERT(11;8;9;10));0)

Die Abarbeitung dieses Terms erfolgt mit einem Operanden(Zahlen)-Stapel und einem Operatoren(Funktionen)-Stapel. (Zur Abkürzung dienen M für „Mittelwert“ und R für „Runden“):

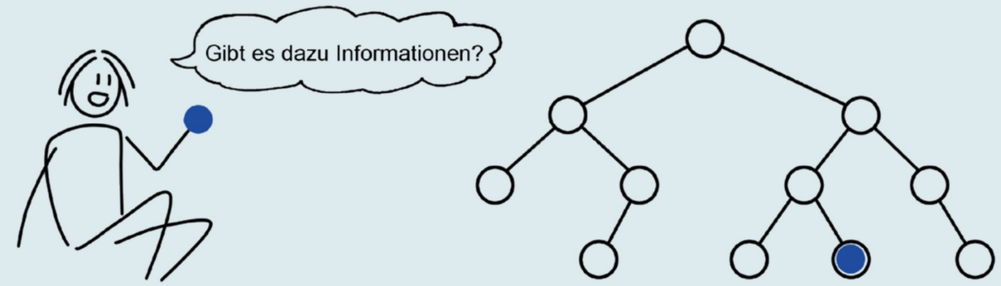


Nach und nach werden Funktionen, Klammern und Zahlen eingelesen und auf den jeweiligen Stapel gelegt. Bei einer schließenden Klammer werden die oberste Funktion vom Operatoren-Stapel und die Zahlen bis zur obersten öffnenden Klammer vom Operanden-Stapel geholt. Die Funktion wird ausgewertet und das Ergebnis wieder auf den Stapel gelegt, bis der Term abgearbeitet ist.

2 Die rekursive Datenstruktur Baum

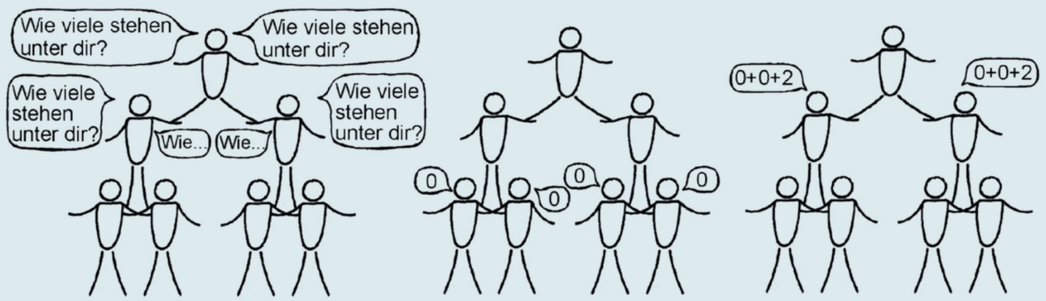
In diesem Kapitel erfahren Sie, ...

... warum die Datenstruktur geordneter Binärbaum eine effektive Suche in umfangreichen Daten ermöglicht.



... wie der Binärbaum als rekursive Datenstruktur umgesetzt werden kann.

... wie rekursive Methoden helfen, mit der Datenstruktur Baum zu arbeiten.



2.1 Effiziente Suchen unterstützen: Datenstruktur Baum

Aylin und Theo spielen folgendes Ratespiel: Aylin denkt sich eine Zahl zwischen 1 und 100 aus. Theo hat maximal sieben Versuche, die Zahl zu erraten. Bei jedem Fehlversuch sagt ihm Aylin, ob seine Zahl zu groß oder zu klein ist.

a Theos Strategie ist es, immer mit seinem Alter (17) zu beginnen und dann in 5er Schritten nach oben bzw. unten zu gehen, um sich so der Zahl anzunähern. Begründen Sie, bei welchen der Zahlen 5, 42, 50 und 90 Theo Erfolg hat.

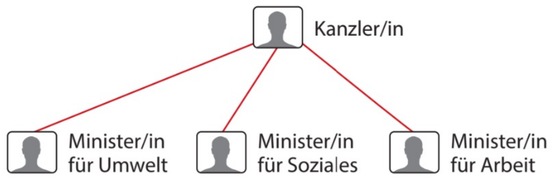
b Entwickeln Sie eine alternative Strategie, die bei allen unter a) genannten Zahlen zum Erfolg führt. Denken Sie daran, dass Theo maximal sieben Versuche hat.

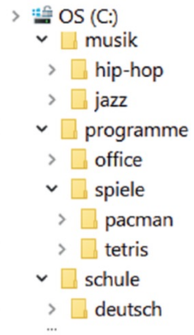
c Für Schnelle: Begründen Sie für Ihre Antwort aus b), ob für alle Zahlen von 1 bis 100 sieben Rateversuche ausreichend sind.

Hierarchische Ordnung – Baumdiagramme

→ Die Richtlinienkompetenz nach Artikel 65 des Grundgesetzes stärkt die Macht des Bundeskanzlers, der in diesem Rahmen etwa Gesetzesvorhaben in den untergeordneten Ressorts anstoßen kann.

Bei der Ordnerstruktur in der Dateiverwaltung, bei der →Richtlinienkompetenz der Bundeskanzlerin oder des Bundeskanzlers oder auch bei Leitungsfunktionen in Organisationen ist eine eindeutige Rangfolge von übergeordneten und untergeordneten Elementen festgelegt. Solche Strukturen nennt man hierarchisch.

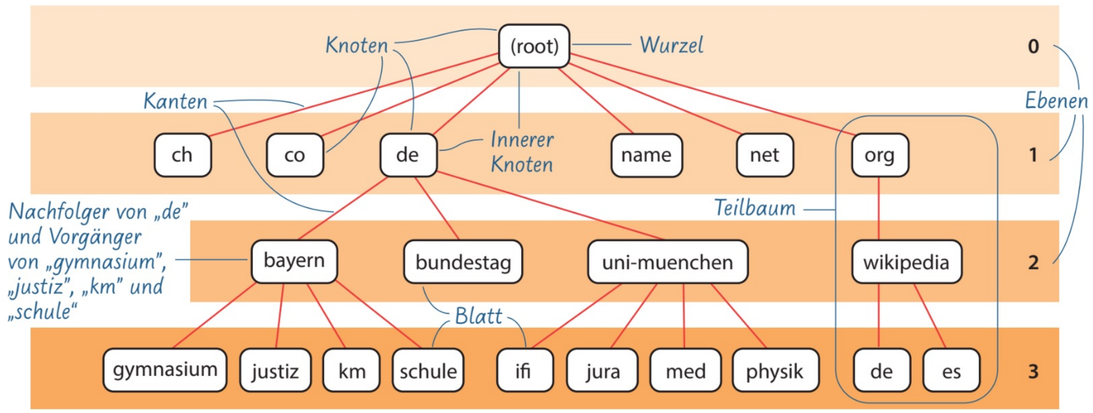




Hierarchische Strukturen lassen sich als Baumdiagramme darstellen, siehe z. B. der Aufbau von URLs unten. Baumdiagramme bestehen aus **Knoten** und **Kanten.** Dabei hat jeder Knoten genau einen Vorgänger, nur die **Wurzel** als „oberster Knoten" hat keinen. Jeder Knoten kann beliebig viele Nachfolger haben. Knoten ohne Nachfolger werden als **Blätter** bezeichnet, alle anderen als **innere Knoten**.

Als einen **Pfad** zwischen zwei Knoten A und B bezeichnet man die Folge der (verschiedenen) Knoten, die aufeinanderfolgend durch Kanten miteinander verbunden sind.

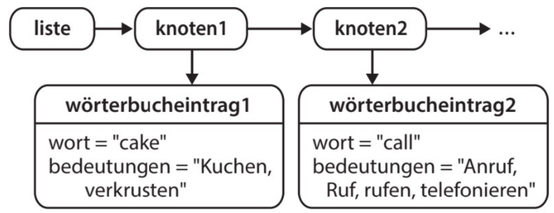
„bundestag → de → bayern → schule" ist also der Pfad von „bundestag" nach „schule".



Die **Tiefe** eines Knotens K ergibt sich aus der Anzahl der Kanten, die beim Durchlaufen des Pfades von der Wurzel zum Knoten K überschritten werden. Alle Knoten mit gleicher Tiefe bilden eine **Ebene**. Die **Höhe** eines Baumes ist die Tiefe des Knotens, der am weitesten von der Wurzel entfernt ist. Der Baum im Beispiel mit den URLs hat die Höhe 3.

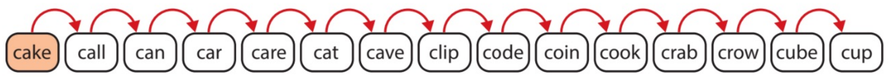
Implementierung eines Wörterbuchs: Prototyp 1 mit der Datenstruktur Liste

Die Vorteile der Baumstruktur für bestimmte Anwendungen zeigen sich sehr gut im Vergleich zur bereits behandelten Datenstruktur „Liste". Daher soll als Beispiel ein simples, digitales englisch-deutsches Wörterbuch zunächst mit Hilfe einer sortierten Liste implementiert werden. Als Vereinfachungen für einen ersten Prototypen wird die Übersetzungsrichtung von Englisch auf Deutsch eingeschränkt, so dass jedem englischen Wort auch mehrere, durch ein Komma getrennte, deutsche Bedeutungen zugeordnet werden können. Die Einträge sind abhängig vom Schlüsselattribut wort →lexikographisch sortiert.



In den folgenden Abbildungen werden Knoten und Wörterbucheinträge verkürzt nur durch das Schlüsselwort dargestellt. Hier ein Ausschnitt der Liste mit 15 aufeinanderfolgenden Einträgen:

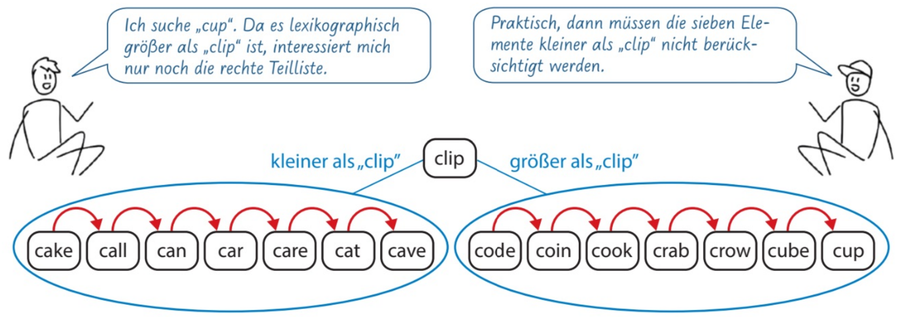
→ alphabetische Ordnung wie bei Einträgen in einem Lexikon



Um in diesem Teil der Liste ein Wort zu finden, benötigt man im besten Fall („cake") einen Vergleich, im schlechtesten Fall („cup") 15 Vergleiche, im Mittel also 8 Vergleiche.

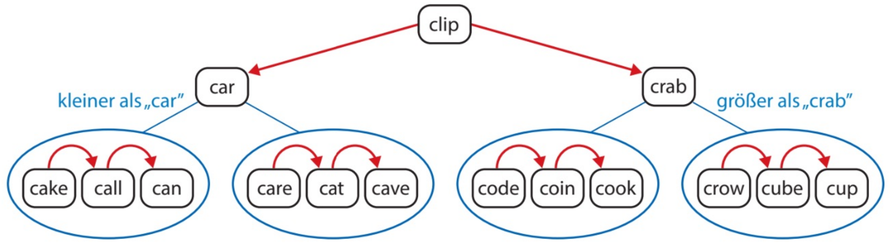
Den Suchaufwand reduzieren

Um im ungünstigsten Fall nicht alle Elemente einer sortierten Liste abfragen zu müssen, sollte die Suche in der Mitte der Liste beginnen. Je nachdem, ob das gesuchte Wort lexikographisch kleiner oder größer als das Wort in der Listenmitte ist, werden anschließend nur die Elemente in den Teillisten links oder rechts vom mittleren Element durchsucht.

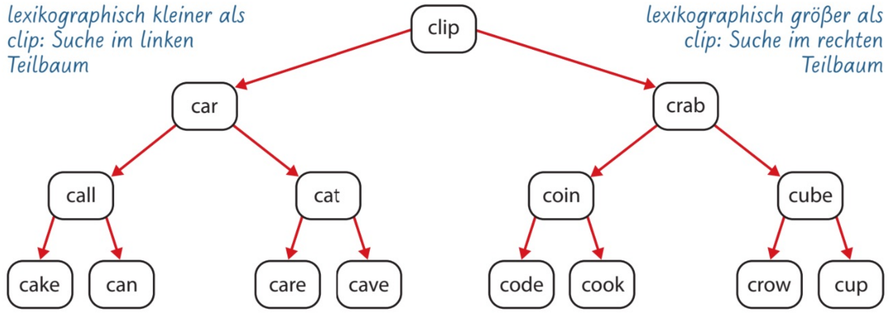


Im besten Fall benötigt man einen Vergleich, im schlechtesten Fall nur noch 8 Vergleiche, im Mittel ca. 4,73 Vergleiche.

Das gleiche Verfahren kann man auch auf die Teillisten anwenden. Die Anzahl der Vergleiche sinkt weiter (Maximal 5, im Mittel 3,53 Vergleiche):

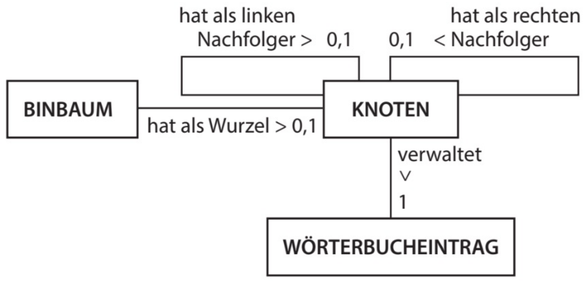


Die Umstrukturierung lässt sich so lange fortführen, bis jede neue Teilliste nur noch aus einem einzigen Element besteht. Im Beispiel verkleinert sich die Anzahl der Vergleichsoperationen für die Suche nach dem Wort „cup" auf maximal 4; im Mittel sind es 3,26 Vergleiche.



Datenstruktur geordneter Binärbaum

Durch die Umstrukturierung im vorigen Abschnitt ist ein Baum mit jeweils maximal zwei Nachfolgern bei jedem Knoten entstanden, ein sogenannter **Binärbaum.**



Bezogen auf die lexikographische Ordnung sind im linken Teilbaum alle Datenelemente kleiner als das Datenelement der Wurzel, im rechten Teilbaum hingegen größer. Man spricht daher von einem **geordneten Binärbaum**.

Bei der Suche in einem Wörterbuch wurde oben die Reduzierung der maximal notwendigen Vergleichsoperationen von 15 auf 4 gezeigt. Bei großen Datenmengen – in Wörterbüchern liegt die Anzahl typischerweise in der Größenordnung von 100 000 – macht sich der geringere Aufwand deutlich bemerkbar:

Datenstruktur mit 100 000 Knoten	maximaler Suchaufwand	mittlerer Suchaufwand
Liste	100 000	50 000,5
optimal geordneter Binärbaum	17	15,61

Die Suche in einem geordneten Binärbaum ist effizienter als in einer sortierten Liste. Die hier vorgestellte Art zu suchen nennt man binäre Suche. Die Suche ist rekursiv, denn es wird immer dasselbe Verfahren angewendet, allerdings auf immer kürzere (Teil-)Bäume, bis die (Teil-)Bäume nur noch aus einem Element bestehen.

Anzahl der Elemente im Binärbaum

In der Tabelle rechts ist die maximale Anzahl der Knoten in einem Binärbaum abhängig von der Anzahl der Ebenen dargestellt. Daraus ergibt sich, dass mit n Ebenen **maximal $2^n - 1$** Knoten gespeichert werden können. Ist diese maximale Knotenzahl erreicht, nennt man den Binärbaum **vollständig**.

Anzahl der Ebenen n	max. Anzahl der Knoten
1	1
2	3
3	7
4	15
…	…
n	2^n-1

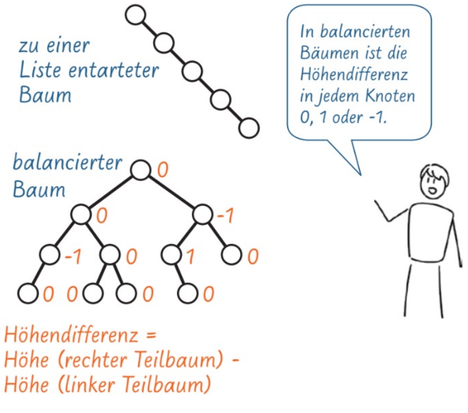
Da nicht jeder Binärbaum vollständig ist, gilt für n Ebenen:

$$\text{Knotenzahl} <= 2^n - 1 \quad | +1$$
$$\text{Knotenzahl} + 1 <= 2^n \quad | \log_2$$
$$\log_2(\text{Knotenzahl} + 1) <= n$$
$$\text{bzw. } n >= \log_2(\text{Knotenzahl} + 1)$$

Für einen Binärbaum mit einer bestimmten Knotenzahl gilt somit: Die Anzahl der Ebenen ist immer mindestens $\log_2$(Knotenzahl + 1) Ebenen.

Effektive Suche nur in balancierten Binärbäumen

Wenn sich die Höhe im linken und rechten Teilbaum eines Knotens stark unterscheidet, ist die Suche nicht effizient. Im Extremfall kann ein Baum auch zu einer Liste **entartet** sein. Ein geringer Suchaufwand ist bei einem **balancierten** Binärbaum gegeben. Dort unterscheiden sich die linken und rechten Teilbäume eines jeden Knotens in der Höhe um maximal eins.



Der **Baum** ist eine Datenstruktur, die aus **Knoten** und **Kanten** besteht. Jeder Knoten außer der **Wurzel** hat genau einen **Vorgänger**. Bei den Knoten wird zwischen **Blättern** und **inneren Knoten** unterschieden. Die **Tiefe** eines Knotens ist ein Maß für die Länge des Pfades bis zur Wurzel. Alle Knoten mit gleicher Tiefe bilden ein **Ebene**. Die **Höhe** eines Baumes ist die Tiefe des Knotens, der am weitesten von der Wurzel entfernt ist.
Ein **Binärbaum** ist ein Baum, bei dem jeder Knoten höchstens zwei Nachfolger hat. In einem Binärbaum mit n Ebenen können **maximal $2^n - 1$ Knoten** gespeichert werden.
Ein **geordneter Binärbaum** ist ein Binärbaum, bei dem im linken Teilbaum jedes Knotens nur kleinere und im rechten Teilbaum nur größere Datenelemente bezüglich einer Ordnungsrelation stehen (oder umgekehrt). So entsteht eine effiziente Datenstruktur für die Suche in großen Datenbeständen, wenn er nicht **entartet**, sondern möglichst **vollständig** oder **balanciert** ist.

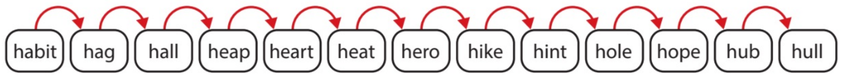
Aufgaben

1 Ratespiel

Im Ratespiel aus der Einstiegsaufgabe wird der mögliche Zahlenbereich von [1;100] auf [1;1000] verzehnfacht. Begründen Sie durch Berechnung der minimalen Ebenen-Anzahl, dass sich die maximale Anzahl der nötigen Rate-Versuche nicht verzehnfacht.

2 Eine Liste umstrukturieren

a Strukturieren Sie ähnlich zu den Abbildungen im Lehrtext die folgende Liste zu einem Binärbaum um.



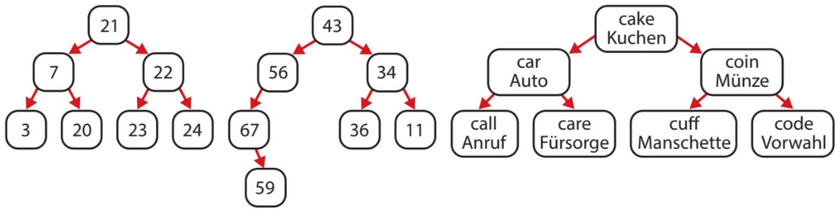
b Beschreiben Sie knapp, an welchen Stellen das Vorgehen „schwieriger" als im Lehrtext ist und wie Sie diese Schwierigkeit lösen.

c Nennen Sie die Anzahl der Vergleiche bei der Suche nach „honest" in Liste und Binärbaum.

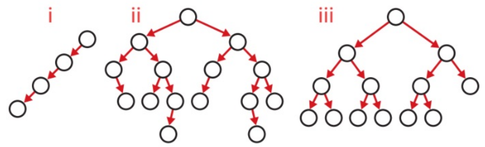
d Führen Sie die interaktiven Übungen in dem Lernpfad zu den ersten beiden Kapiteln durch.

3 Besondere Binärbäume oder nicht?

a Entscheiden Sie begründet, ob die folgenden Bäume geordnete Binärbäume sind.

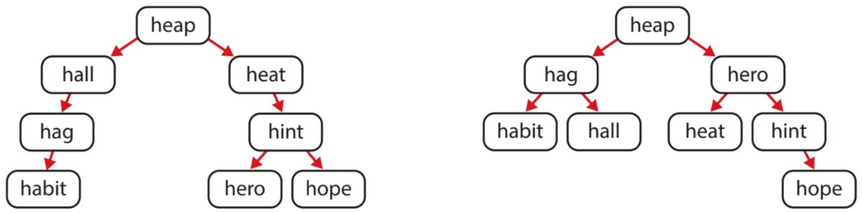


b Entscheiden Sie begründet, welche der Bäume rechts und aus a) jeweils entartet, vollständig bzw. balanciert sind.



4 Wörterbuch als geordneter Binärbaum

Gegeben sind folgende zwei Suchbäume für die gleichen Einträge in einem Wörterbuch.



a Bestimmen Sie jeweils die Anzahl der inneren Knoten, Blätter und Ebenen.

b Nennen Sie die Tiefe der Knoten heap und hope.

c Nennen Sie je einen Knoten, der Wurzel, innerer Knoten bzw. Blatt ist.

d Erläutern Sie, warum beim Suchaufwand nicht nur die Anzahl der Ebenen, sondern auch die Anzahl der Knoten in der untersten Ebene eine Rolle spielt.

e Notieren Sie die Pfade von den Knoten hag zu heat in beiden Bäumen.

f Zeichnen Sie einen geordneten Binärbaum mit den gleichen Einträgen wie oben, in dem hope aber die Tiefe 5 hat.

5 Richtig oder falsch

Entscheiden Sie zu jeder Aussage, ob sie richtig oder falsch ist. Begründen Sie Ihre Entscheidungen und berichtigen Sie bei falschen Aussagen.

a Jeder Baum ist eine Liste.

b Jeder Knoten in einem Baum hat genau einen übergeordneten Knoten.

c Die Anzahl der inneren Knoten ist immer größer als die Anzahl der Blätter.

d Jeder Baum mit k Knoten hat genau k-2 Kanten.

e Jeder Baum ist ein Graph.

6 Knotenanzahl pro Ebene bei Binärbäumen

a Erstellen Sie eine Tabelle mit zwei Spalten. Tragen Sie in der linken Spalte die Nummer der Ebene ein (0, 1, 2, 3,…, n) und in der rechten Spalte die maximale Anzahl der Knoten in dieser Ebene.

b Begründen Sie, warum es sinnvoll ist der obersten Ebene die Zahl 0 und nicht die Zahl 1 zuzuordnen.

7 WWW-Adressen

Gegeben ist folgende Webadresse:
stadtrat.infohausen.de/mitglieder/wahlperiode2025/bayer.html

a Erklären Sie knapp, warum in der URL zwei unterschiedliche Hierarchien enthalten sind.

b Geben Sie zwei weitere URLs an, die auch eine Unterdomäne enthalten. Zeichnen Sie von den drei Domänen die Domänenstruktur als Baum.

c Begründen Sie, warum bei Internet-Domänen eine hierarchische Struktur sinnvoll ist.

d Für Schnelle: Erklären Sie, warum auch wikipedia.de zu einem erfolgreichen Seitenaufbau führt, obwohl die Adresse nach der Abbildung im Lehrtext de.wikipedia.org heißen müsste. Recherchieren Sie, falls nötig.

8 Suchaufwand

In einem noch recht jungen sozialen Netzwerk sind eine halbe Millionen User registriert. Zur Neuanmeldung werden alle User in einem geordneten Binärbaum in den Arbeitsspeicher des Servers geladen, um schon während der Eingabe überprüfen zu können, ob der Benutzername existiert.

a Schätzen Sie ab, wie lange das Überprüfen eines Benutzernamens maximal dauert. Gehen Sie davon aus, dass der Binärbaum ausgeglichen ist und dass ein Vergleich $3*10^{-6}$ s dauert.

b Begründen Sie, warum die in a) berechnete Zeit oft unterschritten wird.

c Erläutern Sie knapp, warum die Datenstruktur Liste in diesem Anwendungsfall für die User nicht akzeptabel ist.

9 Kinoverwaltung Teil 1 (aus Abitur 2020)

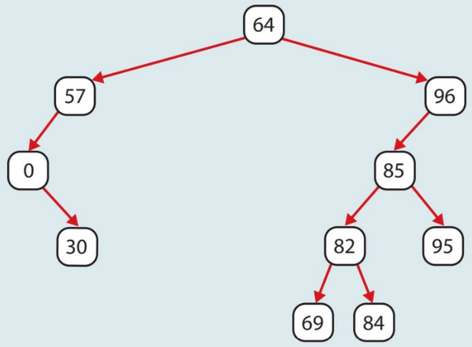
Eine Kinosoftware verwaltet die Kundendaten in einem geordneten Binärbaum, wobei als Schlüssel die Kundennummer dient, die sich aus den zwei Anfangsbuchstaben sowohl des Vor- als auch des Nachnamens sowie einer fortlaufenden Zahl zusammensetzt. Die Sortierung erfolgt lexikographisch. Die Kunden […] sollen in einen […] Binärbaum so eingefügt werden, dass der entstehende Binärbaum möglichst wenig Ebenen besitzt: […]

a Gehen Sie nun davon aus, dass der Binärbaum aus neun Ebenen besteht und 295 Kunden enthält. Berechnen Sie unter der Bedingung, dass die Anzahl der Ebenen gleich bleiben soll, wie viele Kunden darin im besten Fall noch Platz finden.

b Geben Sie zusätzlich für die oben beschriebene Bedingung einen Term an, der in Abhängigkeit von der Ebenenanzahl n und der Anzahl k der gespeicherten Kunden die Anzahl der freien Plätze beschreibt.

2.2 Suchen und Einfügen: Rekursion im geordneten Binärbaum

Um einen geordneten Binärbaum zu erzeugen, muss man neue Elemente in einen (am Anfang leeren) Baum richtig einfügen. Der abgebildete Binärbaum enthält schon einige Datenelemente. Das Schlüsselattribut, nach dem geordnet wird, besteht aus den angegebenen Zahlen.



a Nennen Sie einen Anwendungszusammenhang für größere Datenmengen, bei denen der Schlüssel eine Zahl ist.

b Starten Sie eine App zum Visualisieren von Methodenaufrufen beim Baum.
Stellen Sie einen schrittweisen Ablauf ein und fügen Sie einen neuen Knoten ein.
Beschreiben Sie auf Basis Ihrer Beobachtungen den Algorithmus zum Einfügen in einem geordneten Binärbaum.

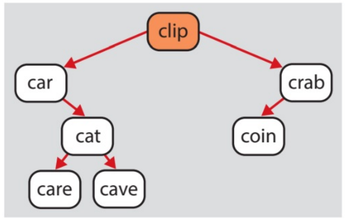
c Im Programm werden zur Veranschaulichung Farben verwendet. Nennen Sie zu jeder Farbe deren Bedeutung im Ablauf beim Einfügen.

Rekursives Einfügen in einen geordneten Binärbaum

Um einen geordneten Binärbaum zur Datenspeicherung nutzen zu können, müssen schrittweise Knoten mit Daten eingefügt werden. Dabei werden Knoten immer an „freien" Stellen eingefügt, d. h. der eingefügte Knoten wird ein Blatt. Beim Einfügen muss darauf geachtet werden, dass die Ordnung des Binärbaums erhalten bleibt. Es ist somit wichtig, den Baum an der richtigen Stelle um einen Knoten zu erweitern. Voraussetzung dafür sind Vergleichsoperationen, die prüfen, ob ein Datenelement größer als ein anderes ist oder ob sie gleich groß sind. Diese werden in einer rekursiven Methode *Einfügen(datenNeu: DATENELEMENT)* genutzt.
Als Beispiel zeigt die folgende Bildsequenz den Ablauf beim Einfügen des Wortes `"cook"` in den Datenbestand eines Wörterbuchs, der in einer Binärbaumstruktur verwaltet wird.

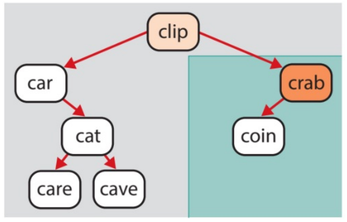
wurzel.Einfügen("cook")
Daten sind gleich? nein
Sind eigene Daten größer als die neuen Daten? nein
Gibt es einen rechten Nachfolger? ja
Beauftrage rechten Nachfolger mit dem Einfügen.

Beschreibung des Ablaufs: Anweisungen, Auswertung von Bedingungen



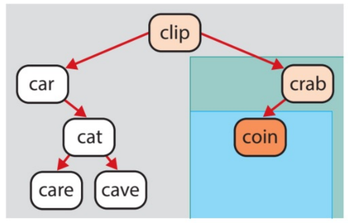
rechter Nachfolger

wurzel.rechterNf.Einfügen("cook")
Daten sind gleich? nein
Sind eigene Daten größer als die neuen Daten? ja
Gibt es einen linken Nachfolger? ja
Beauftrage linken Nachfolger mit dem Einfügen.

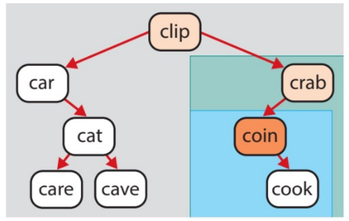


linker Nachfolger

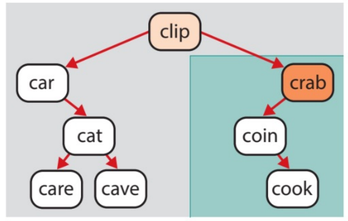
wurzel.rechterNf.linkerNf.Einfügen("cook")
Daten sind gleich? nein
Sind eigene Daten größer als die neuen Daten? nein
Gibt es einen rechten Nachfolger? nein



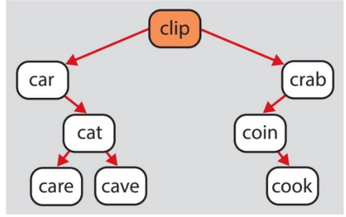
Erzeuge einen neuen Knoten mit Referenz auf das neue Datenelement mit dem Schlüssel `"cook"` und weise ihn als rechten Nachfolger zu.
//fertig



//fertig



//fertig



Der Ablauf zeigt, dass beim Aufruf der Methode *Einfügen* eines Knotens drei Vergleichsfälle unterschieden werden müssen:

(1) Die Datenelemente sind gleich: Eine Abbruchbedingung ist erfüllt, da das Datenelement schon vorhanden ist und es hier nicht erlaubt ist, dass ein Schlüssel mehrfach vorkommt. Das Einfügen wird beendet.

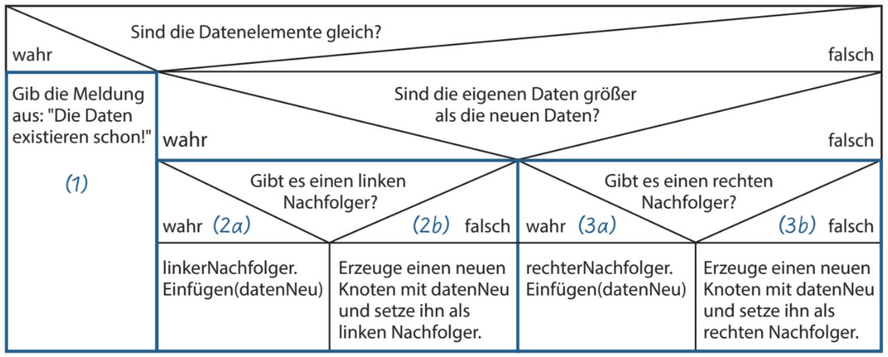
(2a) Das eigene Datenelement ist größer als die neu einzufügenden Daten: Der Einfügeauftrag wird durch einen rekursiven Aufruf an den linken Nachfolger weitergeleitet.

(3a) Das eigene Datenelement ist kleiner als die neu einzufügenden Daten: Der Einfügeauftrag wird durch einen rekursiven Aufruf an den rechten Nachfolger weitergeleitet.

Bevor jedoch ein Nachfolger mit dem Einfügen beauftragt wird, muss überprüft werden, ob er vorhanden ist. Wenn es den betreffenden Nachfolger nicht gibt, ist eine Abbruchbedingung der Rekursion erfüllt: Es wird ein neuer Knoten mit dem einzufügenden Datenelement erzeugt und als neuer Nachfolger in den Baum „eingehängt" (2b) bzw. (3b).

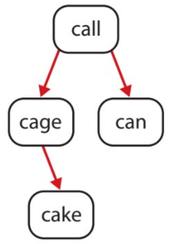
Rekursion macht es mir hier wieder leicht: Egal, welche Daten schon im Baum enthalten sind, genau die richtigen Knoten werden beauftragt.

Einfügen(datenNeu)



Suche mit rekursiven Aufrufen

Auch bei der Suche gibt es die gleichen drei Fälle. Ausgangspunkt des folgenden Beispiels ist jeweils ein Aufruf der Methode *Suchen(suchschlüssel)* bei der Wurzel im Baum rechts.



(1) Suche das Datenelement mit dem Schlüssel `"call"`: Die Schlüssel der Datenelemente sind gleich. Das Suchen wird beendet und die Daten zurückgegeben.

(2a) Suche das Datenelement mit dem Schlüssel„cake“: Der eigene Schlüssel `"call"` ist größer als `"cake"`. Der Suchauftrag wird durch einen rekursiven Aufruf an den linken Nachfolger weitergeleitet.

(3a) Suche das Datenelement mit dem Schlüssel `"can"`: Der eigene Schlüssel `"call"` ist kleiner als `"can"`. Der Suchauftrag wird durch einen rekursiven Aufruf an den rechten Nachfolger weitergeleitet.

Gibt es den Nachfolger nicht, der mit dem Suchen beauftragt werden soll, ist eine Abbruchbedingung der Rekursion erfüllt und es wird rückgemeldet, dass das entsprechende Datenelement nicht im Baum enthalten ist (2b) bzw. (3b).

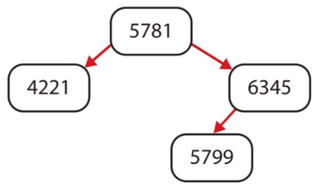
Wichtige Methoden der Datenstruktur geordneter Binärbaum sind das **Suchen** und das **Einfügen** von Datenelementen. Beide Methoden arbeiten rekursiv. Eine Abbruchbedingung ist erfüllt, wenn das Datenelement gefunden wurde oder kein Nachfolger mehr existiert. Ansonsten erfolgt ein rekursiver Aufruf.

Aufgaben

1 Entstehung eines Baumes

a Geben Sie jede mögliche Reihenfolge der Elemente des geordneten Binärbaums rechts an, in der sie eingefügt worden sein könnten.

b Zeichnen Sie den Baum, der entsteht, wenn zusätzlich die Elemente mit den Zahlen 6300, 5888 und 5788 nacheinander eingefügt werden.



2 Rekursive Methode Suchen

a Erarbeiten Sie sich anhand des Lernpfads den Algorithmus zum Suchen im Binärbaum. Fassen Sie wesentliche Ergebnisse knapp zusammen. Wie bei der Methode *Einfügen* im Lehrtext ist Pseudoquelltext ausreichend, die Implementierung erfolgt im nächsten Kapitel.

b Skizzieren Sie ein Struktogramm zur Methode *Suchen(suchschlüssel)*. Markieren Sie mit Farben Abbruchbedingungen, rekursive Aufrufe sowie Rekursionsenden.

3 Ablauf rekursiver Aufrufe

Beschreiben Sie für jeden Methodenaufruf den daraus folgenden Ablauf. Verwenden Sie dabei den Baum und die Schreibweise aus der Abbildung im Lehrtext.

a wurzel.Suchen("cube")
b wurzel.Suchen("coin")
c wurzel.Einfügen("cube")
d wurzel.Einfügen("care")

4 Netzwerkgeräte

In einem Netzwerk eines großen Unternehmens werden alle Geräte durch eine Buchstaben-Zahlen-Kombination eindeutig identifiziert: R steht für Router, N für Notebook, P für Drucker. Die Daten zu den Geräten werden in einem geordneten Binärbaum verwaltet, der lexikographisch sortiert ist.

a Zeichnen Sie den Baum, der sich ergibt, wenn Daten mit den folgenden Schlüsseln in der angegebenen Reihenfolge in einen leeren Binärbaum eingefügt werden.
N202, R273, N304, P114, N111, R212, N264

b Geben Sie die Tiefe des Baums aus a) an und berechnen Sie, wie viele Ebenen für 2000 Geräte mindestens nötig sind.

c Bewerten Sie das Vorgehen, alle Geräte geordnet nach Typ und Nummer einzufügen.

5 Mitgliederverwaltung in einem sozialen Netzwerk (aus Abitur 2018)

Um eine Alternative zu bestehenden sozialen Netzwerken zu bieten, entwickelt ein Team […] das Netzwerk Abi18Net. […] Um alle Mitglieder zu verwalten, wird […] ein geordneter Binärbaum verwendet. Als Schlüssel dient dabei eine für jedes Mitglied eindeutige Mitgliedsnummer.

a Es wird zunächst vorgeschlagen, die Mitgliedsnummern in aufsteigender Reihenfolge gemäß der Anmeldung zu vergeben. Nehmen Sie zu diesem Vorschlag im Kontext der vorgesehenen Datenverwaltung Stellung.

b Für Schnelle: Die Softwareentwickler beschließen abweichend vom Vorschlag aus a), dass die Mitgliedsnummern zufällige, noch nicht vergebene natürliche Zahlen sein sollen. Beschreiben Sie einen Algorithmus, der eine derartige Mitgliedsnummer findet, wenn Methoden zum Erzeugen einer Zufallszahl und zum Suchen einer Mitgliedsnummer im Binärbaum bereits zur Verfügung stehen.

c Zeichnen Sie den entstehenden Binärbaum, wenn die Mitglieder mit den in Klammern angegebenen Mitgliedsnummern in dieser Reihenfolge in einen zunächst leeren Binärbaum eingefügt werden: Karla (3257), Alexej (66013), Bert (116), Dagmar (5485), Holger (2485), Jörg (833271), Paula (73680). Vereinfachend kann jeder Knoten durch die zugehörige Mitgliedsnummer bezeichnet werden.

2.3 Entwurfsmuster nutzen: Binärbaum implementieren

Moritz hat eine Nutzerverwaltung als geordneten Binärbaum umgesetzt.

a Analysieren Sie die Struktur seines Programms und fassen Sie Ihr Ergebnis als Klassendiagramm zusammen. Bei der Klasse USER ist es ausreichend, das Schlüsselattribut zu notieren, Methoden können Sie weglassen.

b Korrigieren Sie den Fehler bei den Referenzattributen in der Klasse KNOTEN.

c Testen Sie unterschiedliche Fälle der Methode *Einfügen* der Klasse KNOTEN. Nutzen Sie dazu die vorhandenen Objekte der Testumgebung im Projekt.

d Alternativ zur Vorlage lässt sich die Datenstruktur Binärbaum auch mit dem Entwurfsmuster Kompositum umsetzen. Zeichnen Sie ein entsprechendes Klassendiagramm.

e Für Schnelle: Beschreiben Sie am Beispiel von d), wie Polymorphismus im Entwurfsmuster genutzt wird.

Mehr Objekte in der Implementierung des Wörterbuchs – Prototyp 2

In den letzten beiden Kapiteln lag der Fokus auf den Daten. Deshalb wurden in den Abbildungen vereinfacht nur die Knoten mit dem Schlüssel des Datenelements (Wörterbucheintrag) beschriftet. Bei der Implementierung muss man beachten, dass es deutlich mehr Objekte gibt:

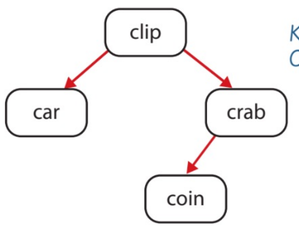
→ 1.2

- Wegen der Trennung von Struktur und Inhalt haben Knoten-Objekte einerseits Referenzen auf Nachfolger, andererseits auch auf die Datenelemente.
- Ein Objekt der Klasse BINBAUM bietet Schnittstellen nach außen an. Es hat eine Referenz auf die Wurzel des Baumes.

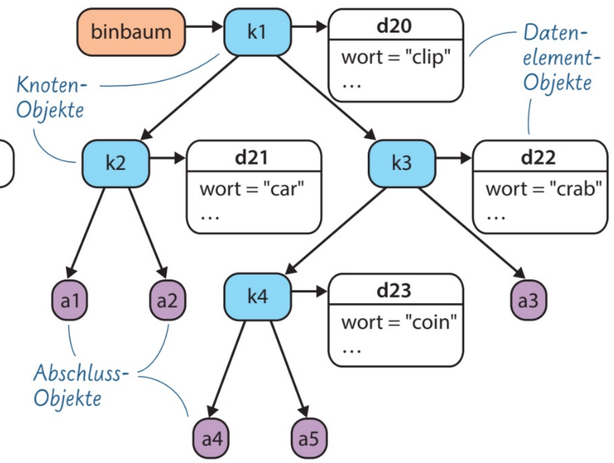
→ 1.5

- Setzt man den geordneten Binärbaum mit dem Entwurfsmuster Kompositum um, ist eine besonders einfache Implementierung der rekursiven Methoden möglich. Bestandteil des Entwurfsmusters ist, dass als Nachfolger eines Knotens nicht nur Knoten-Objekte, sondern auch Abschluss-Objekte referenziert werden können.

vereinfachtes Objektdiagramm ohne Knoten und Abschlüsse

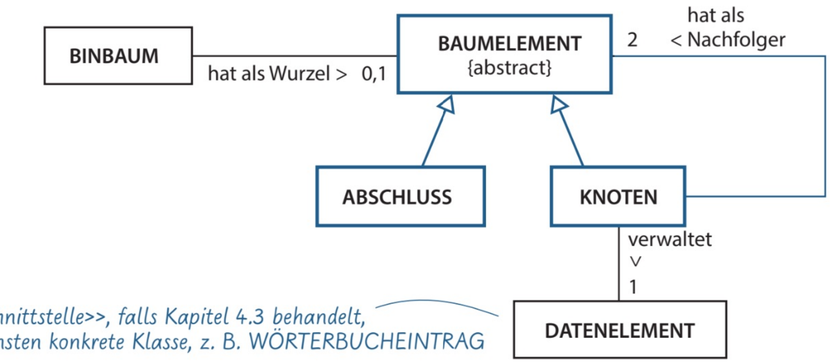


ausführliches Objektdiagramm zur Planung der Implementierung mit Kompositum



Grundstruktur eines geordneten Binärbaumes

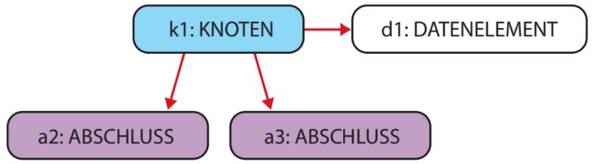
Das Klassendiagramm der Datenstruktur Liste mit Entwurfsmuster Kompositum und Trennung von Struktur und Inhalt lässt sich einfach auf den Binärbaum übertragen. Neben sinnvollen Namensanpassungen ändert sich nur die Kardinalität in der rekursiven Beziehung zwischen BAUMELEMENT und KNOTEN: Jeder Knoten hat genau zwei Nachfolger, nicht genau einen wie bei der Liste. Das Entwurfsmuster Kompositum ist im Klassendiagramm durch eine blaue Linienfarbe hervorgehoben.



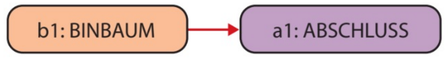
Implementierung des Binärbaums – Teil 1: Attribute und Konstruktoren

Im ersten Schritt der Umsetzung werden nur die Attribute und Konstruktoren betrachtet. Folgendes schrittweises Vorgehen ist sinnvoll:

- Die Klasse BAUMELEMENT hat keine Attribute. Somit müssen im Konstruktor keine Anweisungen ausgeführt werden, der Rumpf ist leer.
- Die Klasse ABSCHLUSS hat keine Attribute. Wiederum müssen im Konstruktor keine Anweisungen ausgeführt werden. Daher muss nur der Konstruktor der Oberklasse aufgerufen werden.
- Die Klasse KNOTEN hat Referenzen auf den rechten und linken Nachfolger und ein Datenelement wie beispielsweise einen Wörterbucheintrag. Die Nachfolger sind vom Datentyp BAUMELEMENT.
  Der Konstruktor soll das Datenelement als Parameter übergeben bekommen. Die Nachfolger sollen für einen neuen Knoten Objekte der Klasse ABSCHLUSS sein, die hier erzeugt werden müssen.



- Die Klasse BINBAUM hat ein Referenzattribut wurzel mit dem Datenyp BAUMELEMENT. Beim Aufruf des Konstruktors wird ein leerer Baum erzeugt, d. h. als Wurzel wird ein Objekt der Klasse ABSCHLUSS erzeugt und referenziert.



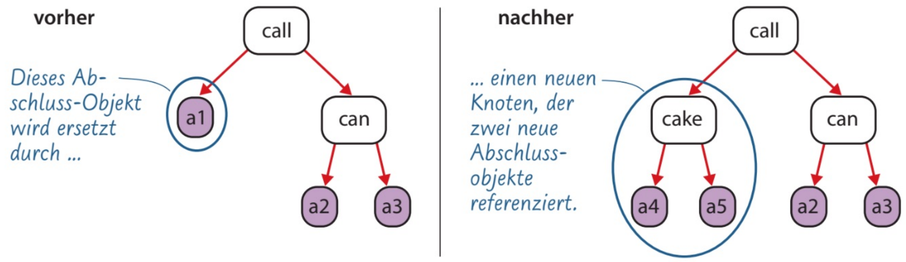
Implementierung des Binärbaums – Teil 2: Einfügen

Bisher kann nur ein leerer Binärbaum erzeugt werden. Über Methodenaufrufe soll nun das Einfügen von Datenelementen ermöglicht werden. Dabei muss die Methode *Einfügen* **jede** der folgenden Anforderungen erfüllen:

(1) Der Binärbaum muss immer geordnet bleiben.
(2) Als Parameter werden Datenelemente (konkret: Wörterbucheinträge o. Ä.) verwendet.
(3) Es ist nicht erlaubt, ein Datenelement einzufügen, dessen Schlüssel im Binärbaum schon enthalten ist. Wird es versucht, muss eine Fehlermeldung ausgegeben werden.

Bei der Umsetzung ist folgendes schrittweises Vorgehen sinnvoll:

- Klasse ABSCHLUSS
  Das Einfügen wird wie bei der Liste mit rekursiven Methoden umgesetzt. Beim geordneten Binärbaum findet das Einfügen immer am Ende eines Astes statt, d. h. bei einem Abschluss-Objekt. Das Abschluss-Objekt wird ersetzt durch einen neuen Knoten. Dieser neue Knoten referenziert das einzufügende Datenelement und zwei Objekte der Klasse ABSCHLUSS. Die folgende Abbildung verdeutlicht dies am Beispiel des Einfügens eines Datenelements mit "cake" als Schlüssel.



Wieder der gleiche Trick wie bei der Liste: Kommunikation mit dem Aufrufer über den Rückgabewert.

Wie schon bei der Liste mit Entwurfsmuster Kompositum ist bei einem Einfügen am Ende das Abschluss-Objekt aktiv. In diesem Beispiel muss a1 den neuen Knoten "cake" erzeugen. Dabei weiß das Objekt "call" beim Beauftragen seines linken Nachfolgers noch nicht, dass es einen neuen linken Nachfolger erhalten wird. Da a1 keine Referenz auf seinen Aufrufer, das Objekt "call", hat, muss es ihn durch einen Rückgabewert über den neuen Nachfolger informieren. Die Methode *Einfügen* der Klasse ABSCHLUSS muss somit als Rückgabe eine Referenz auf ein Baumelement haben.

Klasse ABSCHLUSS
Methode Einfügen(datenNeu: DATENELEMENT) -> BAUMELEMENT

```
return new KNOTEN(datenNeu)
```

Super, wie kurz und übersichtlich die Methodenrümpfe durch das Entwurfsmuster Kompositum sind.

- Da beim Entwurfsmuster Kompositum Abschlüsse und Knoten Unterklassen von BAUMELEMENT sind, wird der Rückgabewert der Methode *Einfügen* auch in der abstrakten Klasse BAUMELEMENT festgelegt …
- … und muss auch in der Klasse KNOTEN so überschrieben werden.

Klasse KNOTEN
Methode Einfügen(datenNeu: DATENELEMENT) -> BAUMELEMENT

```
falls daten.IstGleich(datenNeu)
dann
  konsole.Ausgeben("Daten schon enthalten!")
sonst
  falls daten.IstGrößerAls(datenNeu)
  dann
    linkerNf = linkerNf.Einfügen(datenNeu)
  sonst
    rechterNf = rechterNf.Einfügen(datenNeu)
  endefalls
endefalls
return selbst
```

Fall 1 – gleiche Datenelemente: Rekursionsende, da es bereits einen Eintrag mit dem gleichen Schlüssel gibt.

Fall 2 – eigenes Datenelement ist größer als datenNeu: rekursiven Aufruf an den linken Nachfolger; Speicherung des Rückgabewerts als linker Nachfolger

Fall 3 – analog zu Fall 2 für eigenes Datenelement ist kleiner als datenNeu

In allen Fällen: Rückgabe einer Referenz auf sich selbst – für den Aufrufer ändert sich der Nachfolger nicht.

- Dank des Entwurfsmusters Kompositum referenziert das Attribut wurzel in der Klasse BINBAUM immer ein Objekt – beim leeren Baum einen Abschluss, sonst den Wurzel-Knoten. Deshalb besteht der Rumpf der Methode *Einfügen* nur aus einer Zeile:

Klasse BINBAUM
Methode Einfügen(datenNeu: DATENELEMENT)

```
wurzel = wurzel.Einfügen(datenNeu)
```

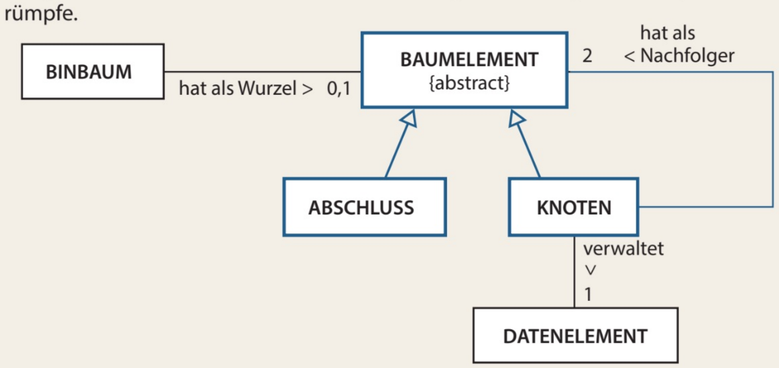
Implementierung des Binärbaums – Teil 3 Suchen

Die Algorithmik beim Suchen ist sehr ähnlich zu der beim Einfügen. Unterschiede gibt es im Methodenkopf: Der Parameter der Suche ist ein Schlüsselwert und die Rückgabe ist das gesuchte Datenelement bzw. null, wenn die Suche erfolglos war.

Implementierungsdetails zum Suchen finden Sie in Aufgabe 2.

Methode Suchen(suchschlüssel: ZEICHENKETTE) -> DATENELEMENT

Bei der Implementierung eines geordneten Binärbaums ist die **Trennung von Struktur und Inhalt** wichtig. Da über zwei Nachfolgerbeziehungen die Ordnung umgesetzt wird, sind in den **rekursiven Methoden Einfügen** und **Suchen** entsprechende Fallunterscheidungen nötig. Der Einsatz des **Entwurfsmusters Kompositum** sorgt für kompakte Methodenrümpfe.



Aufgaben

1 Rollenspiel

Planen Sie ein Rollenspiel zur Veranschaulichung der Implementierung der Methode *Einfügen* mit Entwurfsmuster Kompositum in einem Wörterbuch, das mit einem Binärbaum umgesetzt ist. Aktuell ist ausschließlich das Wort „clip" enthalten.

- **a** Nennen Sie die Anzahl der Objekte inklusive der Klassenzugehörigkeit, die in der Startsituation von Schülerinnen und Schülern als Rolle übernommen werden müssen.
- **b** Nennen Sie die Anzahl der Objekte, die bei einem Aufruf der Methode *Einfügen* hinzukommen.
- **c** Führen Sie das Rollenspiel für das Einfügen zweier unterschiedlicher Wörter durch. Achten Sie bei der Wortauswahl auf unterschiedliche Fälle im Einfügevorgang.

2 Entwurfsmuster Kompositum

Vergleichen Sie die folgenden beiden Umsetzungsvarianten der Methode *Suchen*.

Binärbaum ohne Entwurfsmuster Kompositum

Klasse KNOTEN
Methode Suchen(suchschlüssel: ZEICHENKETTE) ->DATENELEMENT

```
falls daten.SchlüsselIstGleich(suchschlüssel)
dann
  return daten                                    (1)
sonst
  falls daten.SchlüsselIstGrößerAls(↵
                                  suchschlüssel)
  dann
   falls linkerNachfolger != null
   dann
    return linkerNachfolger.↵                     (2)
                          Suchen(suchschlüssel)
   sonst
    return null                                   (3)
   endefalls
  sonst
   falls rechterNachfolger != null
   dann
    return rechterNachfolger.↵                    (4)
                          Suchen(suchschlüssel)
   sonst
    return null                                   (5)
   endefalls
  endefalls
endefalls
```

Binärbaum mit Entwurfsmuster Kompositum

Klasse KNOTEN
Methode Suchen(suchschlüssel: ZEICHENKETTE) ->DATENELEMENT

```
falls daten.SchlüsselIstGleich(suchschlüssel)
dann
  return daten                                    (A)
sonst
  falls daten.SchlüsselIstGrößerAls(↵
                                  suchschlüssel)
  dann
   return linkerNachfolger.↵                      (B)
                          Suchen(suchschlüssel)
  sonst
   return rechterNachfolger.↵                     (C)
                          Suchen(suchschlüssel)
  endefalls
endefalls
```

Klasse ABSCHLUSS
Methode Suchen(suchschlüssel: ZEICHENKETTE) ->DATENELEMENT

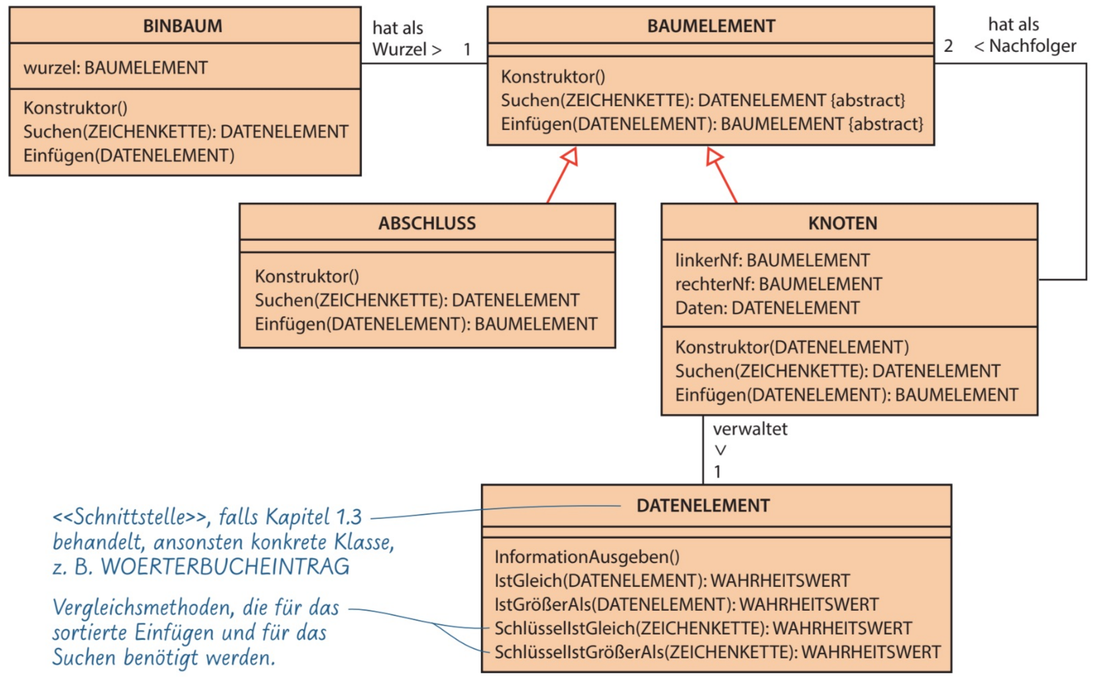
```
return null                                       (D)
```

- **a** Ordnen Sie die zueinander passenden gekennzeichneten Bereiche zu und geben Sie jeweils an, ob es sich um ein Rekursionsende oder einen rekursiven Aufruf handelt.
- **b** Geben Sie für beide Varianten die Abbruchbedingungen an.
- **c** Begründen Sie, warum es in der Variante mit Kompositum zwei Fälle weniger gibt.
- **d** Stellen Sie die Quelltexte der Methode *Suchen* der Klasse BINBAUM in den beiden Variationen mit bzw. ohne Entwurfsmuster Kompositum gegenüber.

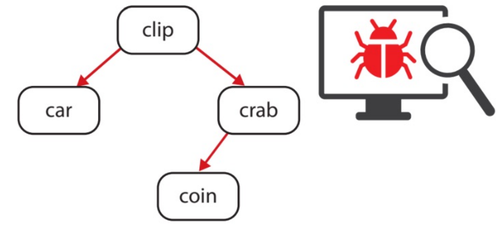
- **e** Vorteile des Entwurfsmusters Kompositum sind einerseits ein kompakter, gut lesbarer Quelltext, andererseits ein jeweils gleiches Verhalten der Objekte. Nennen Sie zwei Nachteile der Umsetzung mit Kompositum.

3 Wörterbuch als geordneter Binärbaum – Teil 1: Struktur, Einfügen und Suchen

- **a** Analysieren Sie in der Klasse WÖRTERBUCHEINTRAG die Umsetzung der Vergleichsmethoden. Erklären Sie sich in Partnerarbeit wechselseitig die Arbeitsweise der Methoden mit einem Datenelement als Eingabe bzw. mit einer Zeichenkette.
- **b** Implementieren Sie einen geordneten Binärbaum mit Wörterbucheinträgen als Datenelemente. Gehen Sie dazu schrittweise wie im Lehrtext beschrieben vor.
  Als Überblick finden Sie unten das erweiterte Klassendiagramm.



- **c** Bewerten Sie an Hand der unterschiedlichen Fälle beim Suchen bzw. Einfügen, ob der Baum rechts zum Testen geeignet ist. Erstellen Sie in Ihrem Projekt einen geeigneten Testbaum, um die Methode *Suchen* einfach testen zu können.
- **d** Erweitern Sie die Klassen um eine Methode *IstVorhanden(suchschlüssel: ZEICHENKETTE)*, welche nur dann WAHR zurückgibt, wenn ein Datenelement mit dem Suchschlüssel enthalten ist.



- **e** Erweitern Sie die Klassen um eine Methode *HöheGeben()*, welche die Höhe des Baums berechnet.
- **f** Für Schnelle: Die Methode *TiefeGeben()* soll abhängig von einem eingegebenen Suchschlüssel die Tiefe des entsprechenden Knotens bestimmen.
  - i Notieren Sie den passenden Methodenkopf.
  - ii Nennen Sie einen geeigneten Rückgabewert für den Fall, dass der Suchschlüssel nicht gefunden wird.
  - iii Setzen Sie die Methode in Ihrem Projekt um.

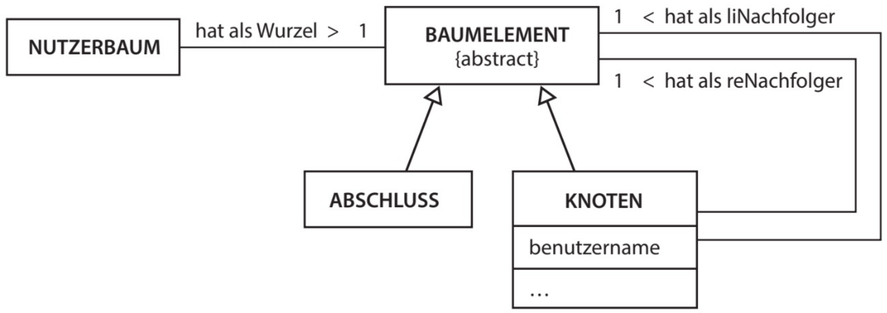
4 Nutzerverwaltung eines Onlinespiels (aus Abitur 2019)

Die persönlichen Daten aller registrierten Nutzer [des Onlinespiels „Achtung Falle“] werden in einer Datenbank gespeichert. Zur Neuanmeldung werden alle vergebenen Benutzernamen in einem lexikographisch geordneten Binärbaum in den Arbeitsspeicher geladen. Schon während der Eingabe des gewünschten Benutzernamens kann bei jedem Tastendruck überprüft werden, ob der im Textfeld stehende Name bereits vergeben ist. […]

a Erläutern Sie, warum in einem für Suchanfragen optimal aufgebauten geordneten Binärbaum nicht nur die Anzahl der Ebenen möglichst gering sein sollte, sondern zusätzlich in der untersten Ebene möglichst wenige Knoten sein sollten. […]

b Bei „Achtung Falle“ sind ungefähr 150 000 Nutzer registriert. Der Binärbaum zum Überprüfen von Benutzernamen hat somit ungefähr 150 000 Einträge. Gehen Sie davon aus, dass nach jeder Aufnahme eines neuen Nutzers dieser Baum gegebenenfalls umstrukturiert wird, sodass er für die Suche immer optimal aufgebaut ist.
Schätzen Sie ab, wie lange das Überprüfen eines eingegebenen Benutzernamens maximal dauert. Nehmen Sie an, dass im Rahmen der Überprüfung ein ganzer Rekursionsschritt mit Vergleich $5 \cdot 10^{-6}$ s benötigt. Beurteilen Sie zudem, ob es eine spürbare Verzögerung für den neuen Nutzer bedeutet, wenn während der Eingabe des gewünschten Benutzernamens nach jedem Tastendruck eine Überprüfung durchgeführt wird.

Der Binärbaum zur Verwaltung der Benutzernamen ist unter Verwendung des Softwaremusters Kompositum gemäß dem abgebildeten Klassendiagramm realisiert.



c Die Klasse NUTZERBAUM soll die Methode *istFrei(benutzername)* besitzen, welche genau dann falsch zurückgibt, wenn das übergebene ZEICHENKETTE-Objekt bereits als Benutzername im Binärbaum vorkommt.
Implementieren Sie diese Methode und alle dazu nötigen Methoden in der Baumstruktur. Verwenden Sie dabei so weit wie möglich das Prinzip der Rekursion und nutzen Sie die Vorteile des geordneten Binärbaums.
Hinweis: Als bereits implementiert vorausgesetzt werden darf eine Methode *vergleichenMit(zk)* der Klasse ZEICHENKETTE, die

- eine negative ganze Zahl zurückgibt, wenn das ausführende Objekt lexikographisch kleiner als die übergebene Zeichenkette zk ist,
- eine positive ganze Zahl zurückgibt, wenn das ausführende Objekt lexikographisch größer als die übergebene Zeichenkette zk ist,
- die Zahl 0 zurückgibt, wenn die Zeichenketten übereinstimmen.

d In der Klassenstruktur des Binärbaums werden die Methoden *m1* und *m2* implementiert. LISTE ist eine Klasse zur Verwaltung einfach verketteter Listen. Die Klasse LISTE verfügt über die Methode *hintenEinfuegen(zk)*, mit der eine übergebene Zeichenkette hinten eingefügt werden kann. Beschreiben Sie, was bei dem Methodenaufruf *m1('s')* zurückgegeben wird.

In der Klasse NUTZERBAUM:

```
methode LISTE m1(ZEICHEN z)
    liste = neues leeres LISTE-Objekt
    wurzel.m2(z, liste)
    gib liste zurück
endemethode
```

In der Klasse KNOTEN:

```
methode m2(ZEICHEN z, LISTE liste)
    ZEICHEN z1 = erstes Zeichen von benutzername
    falls z alphabetisch vor z1 steht
    dann
      liNachfolger.m2(z, liste)
    sonst
      falls z alphabetisch nach z1 steht
      dann
         reNachfolger.m2(z, liste)
      sonst
         liNachfolger.m2(z, liste)
         liste.hintenEinfuegen(benutzername)
         reNachfolger.m2(z, liste)
      endefalls
    endefalls
endemethode
```

5 Nutzerverwaltung optimieren

a Bei der Einstiegsaufgabe wird als Schlüssel für das Einordnen der Nachname verwendet. Beschreiben Sie knapp, warum dies zu Problemen führen kann.

b Überlegen Sie sich eine Strategie, die Problematik aus a) zu vermeiden. Optimieren Sie entsprechend das Projekt der Einstiegsaufgabe.

c Optimieren Sie das Projekt der Einstiegsaufgabe weiter, indem Sie eine Methode *Suchen* ergänzen und auf eine Implementierung mit dem Entwurfsmuster Kompositum umstellen.

6 Kinoverwaltung Teil 2 (aus Abitur 2020)

Bei der Implementierung des geordneten Binärbaums kommt das Entwurfsmuster Kompositum zum Einsatz.
Beschreiben Sie in Worten den rekursiven Algorithmus, den ein Knoten zum Einfügen eines neuen Kunden ausführt. Gehen Sie zudem kurz auf die Rolle des Abschlusses ein.

2.4 Binärbäume durchlaufen: Information aller Daten abfragen

Die Firmenleitung möchte eine Liste aller ihrer Nutzer ihrer neuen App haben. Pia hat dazu eine Methode *InorderAusgeben* geschrieben. Diese durchläuft alle Knoten des Baums und gibt die Namen aller gespeicherten Personen aus.

a Analysieren Sie zuerst die Testumgebung und zeichnen Sie den Baum, der dadurch erzeugt wird. Eine vereinfachte Darstellung nur mit den Schlüsselwerten ist ausreichend.

b Rufen Sie nun die Methode *InorderAusgeben* auf und ordnen Sie jedem Knoten in a) durch eine Zahl zu, ob seine Daten als erstes, zweites, drittes usw. ausgegeben wurden.

c Analysieren Sie die Methode *InorderAusgeben*. Begründen Sie, warum im Gegensatz zu den Methoden *Suchen* und *Einfügen* der Klasse KNOTEN keine bedingte Anweisung nötig ist. Erklären Sie auch, warum der Methodenname mit der Vorsilbe „in" beginnt.

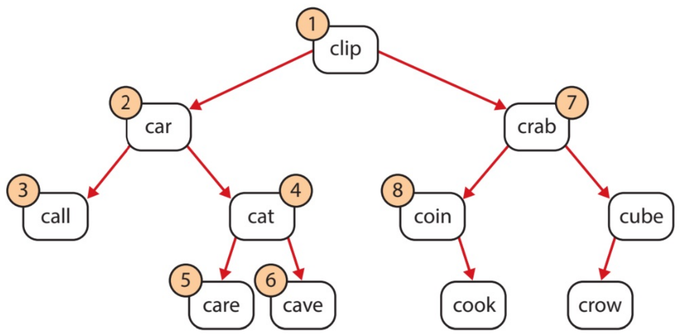
d Die Methode *PreorderAusgeben* der Klasse KNOTEN unterscheidet sich von *InorderAusgeben* passend zur Bedeutung „vor" der Vorsilbe pre. Erschließen Sie sich die Reihenfolge der drei Methodenaufrufe im Rumpf von *PreorderAusgeben* und nennen diese.

e Notieren Sie mit einer anderen Farbe als in b) die Reihenfolge der Preorder-Ausgabe bei dem Baum aus a).

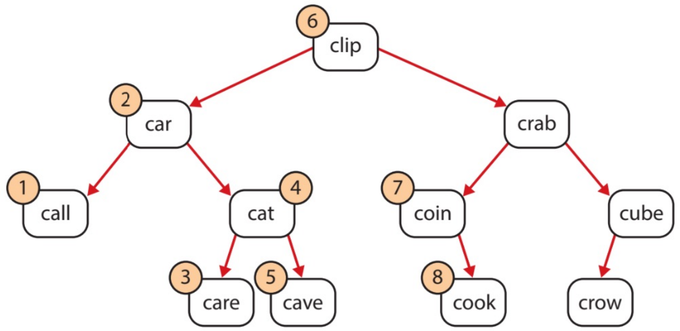
Bäume unterschiedlich traversieren

Häufig möchte man einen Überblick über die Gesamtheit von Daten, auch wenn sie in einem Binärbaum gespeichert sind, beispielsweise alle Nutzer als sortierte Liste oder die hierarchische Ordnerstruktur in der Dateiverwaltung. Die folgenden Abbildungen zeigen zwei unterschiedliche Ausgaben unserer Wörterbucheinträge:

```
1  clip
2     car
3        call
4        cat
5           care
6           cave
7     crab
8        coin
…
```




```
1  call
2  car
3  care
4  cat
5  cave
6  clip
7  coin
8  cook
…
```



Bei der Ausgabe der Daten werden alle Knoten eines Baums erreicht. Man spricht auch von einer Traversierung (franz. Durchquerung) des Baums.

Die Angabe der Durchlaufposition (Nummern) in den Abbildungen oben verdeutlicht, dass die Daten der Baumelemente in den beiden Varianten in unterschiedlichen Reihenfolgen ausgegeben werden.

Sortierte Ausgabe mithilfe eines Inorder-Durchlaufs

Die Ausgabe der Baumelemente mithilfe eines **Inorder-Durchlaufs** ergibt eine sortierte Liste (untere Abbildung auf der Vorseite). Sie hat in der Klasse KNOTEN folgende drei Bestandteile:

Klasse KNOTEN
Methode InorderAusgeben()

```
linkerNf.InorderAusgeben()
daten.InformationAusgeben()
rechterNf.InorderAusgeben()
```

Rekursiver Aufruf der Methode des linken Nachfolgers

Ausgabe der eigenen Daten

Rekursiver Aufruf der Methode des rechten Nachfolgers

Objekte der Klasse ABSCHLUSS haben weder Daten noch Nachfolger. Deshalb müssen Sie beim Aufruf dieser Methode nichts tun, der Methodenrumpf ist leer.
In der Klasse BINBAUM wird nur die Methode *InorderAusgeben()* der Wurzel aufgerufen.

Klasse ABSCHLUSS
Methode InorderAusgeben()

```
// tue nichts
```

Klasse BINBAUM
Methode InorderAusgeben()

```
wurzel.InorderAusgeben()
```

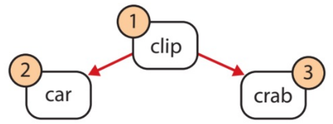
Anwendungen des Preorder-Durchlaufs

Beim →**Preorder-Durchlauf** werden vor den rekursiven Aufrufen die eigenen Daten ausgegeben.

→ lat. prae-: vor (als Vorsilbe)

Klasse KNOTEN
Methode PreorderAusgeben()

```
daten.InformationAusgeben()
linkerNf.PreorderAusgeben()
rechterNf.PreorderAusgeben()
```



Die ausgegebene Liste – im Beispiel clip, car, crab – eignet sich zum sequenziellen Abspeichern von Bäumen: Denn werden die Elemente nacheinander in dieser Reihenfolge wieder eingefügt, wird der ursprüngliche Baum wieder aufgebaut.
Wird beim Preorder-Durchlauf noch eine Einrücktiefe passend zur Tiefe des Knotens ergänzt, veranschaulicht die Ausgabe die hierarchische Struktur. Da kein Knoten seine eigene Tiefe kennt, die Ausgabe aber von den Knoten ausgelöst wird, muss ihnen diese Information durch einen Eingabewert mitgeteilt werden. Bei jedem Aufruf wird tiefe um 1 erhöht.

Einrücktiefe aus Leerzeichen aufbauen

Klasse KNOTEN
Methode StrukturAusgeben(tiefe: GANZZAHL)

```
Erzeuge Zeichenkette s aus 4*tiefe Leerzeichen
Konsole.Ausgeben(s+daten.SchluesselAlsStringGeben())
linkerNf.StrukturAusgeben(tiefe +1)
rechterNf.StrukturAusgeben(tiefe +1)
```

Datenelemente benötigen eine zusätzliche Methode

bei jedem rekursiven Aufruf tiefe um 1 erhöhen

Klasse BINBAUM
Methode StrukturAusgeben()

```
wurzel.StrukturAusgeben(0)
```

Eingabewert 0, weil die Wurzel die Tiefe 0 hat

Das Implementieren der Postorder-Ausgabe ist Thema der Aufgabe 3.

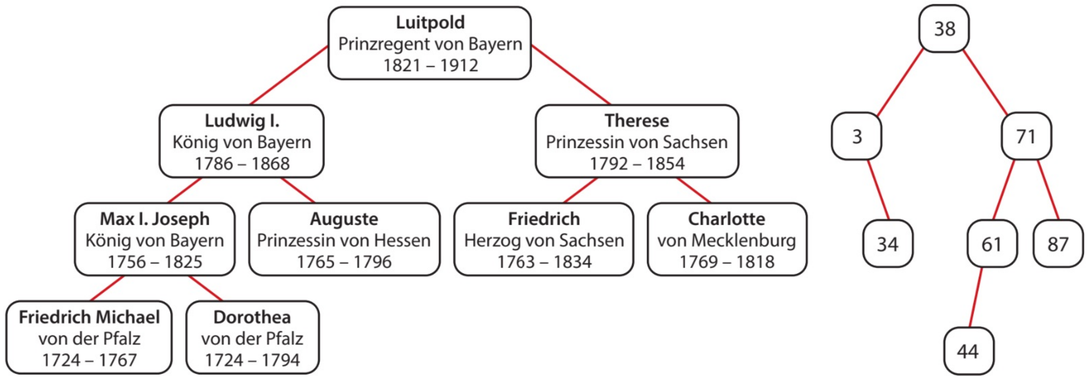
Neben dem Preorder- und Inorder- ist eine weitere Möglichkeit der **Postorder-Durchlauf**, bei dem die Ausgabe der eigenen Daten nach den beiden rekursiven Aufrufen stattfindet.

Sowohl über den **Preorder-**, den **Inorder-** als auch den **Postorder-Durchlauf** werden alle Knoten eines Binärbaums erreicht. Die Durchlaufarten (Traversierungen) unterscheiden sich darin, ob der eigene Inhalt **vor**, **zwischen** oder **nach** den rekursiven Aufrufen an den linken und rechten Nachfolger ausgegeben werden.
Typische Anwendungen für den Preorder-Durchlauf sind eine Ausgabe der hierarchischen Struktur bzw. das sequenzielle Abspeichern eines Baumes. Der Inorder-Durchlauf wird eingesetzt, um die Daten sortiert auszugeben.

Aufgaben

1 Binärbäume durchlaufen

a Nennen Sie die Ausgabe durch einen Preorder- bzw. Inorder-Durchlauf für den Stammbaum und den Baum mit Identifikationsnummern.



b Zeichnen Sie zwei Bäume nach folgenden Vorgaben: Nehmen Sie die beiden Ausgaben des Preorder- und Inorder-Durchlaufs (vgl. Teilaufgabe a) für den Baum mit den Identifikationsnummern und fügen Sie die Elemente in dieser Reihenfolge jeweils in einen leeren Baum ein.
Erklären Sie knapp, warum das Ergebnis zwei sehr unterschiedliche Bäume sind.

c Begründen Sie knapp, welche Durchlaufart zum Speichern eines Baumes genutzt werden kann.

2 Visualisierung des Ablaufs verschiedener Durchlaufarten

Experimentieren Sie mit unterschiedlichen Methoden für den Baumdurchlauf. Erklären Sie sich in Partnerarbeit wechselseitig die Ergebnisse verschiedener Durchlaufarten.

3 Postorder-Durchlauf

Neben dem Preorder- und Inorder-Durchlauf gibt es noch einen Postorder-Durchlauf.

a Überlegen Sie anhand des Namens, wie dieser Durchlauf arbeitet, und beschreiben Sie Ihr Ergebnis knapp.

b Überprüfen Sie Ihr Ergebnis aus a) anhand von zwei Internet-Quellen.

c Nennen Sie für die beiden Bäume aus Aufgabe 1 das Ergebnis eines Postorder-Durchlaufs.

4 Wörterbuch als geordneter Binärbaum – Teil2: Baumdurchläufe

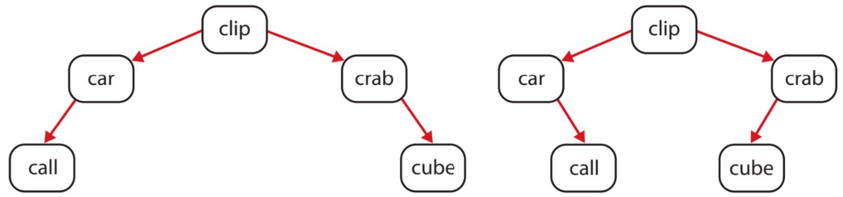
Nehmen Sie als Ausgangsbasis ihr Wörterbuch-Projekt aus dem letzten Kapitel.

a Ergänzen Sie in den Klassen des Entwurfsmusters Kompositum und der Klasse BINBAUM die Methoden *InorderAusgeben, PreorderAusgeben* und *PostorderAusgeben*. Testen Sie mit einem geeigneten Beispielbaum!

b Ergänzen Sie in der Klasse WOERTERBUCHEINTRAG (ggf. auch DATENELEMENT falls Kapitel 1.3 behandelt wurde) die Methode *SchlüsselAlsStringGeben*, die als Rückgabewert den Schlüssel als Zeichenkette ausgibt.

c Implementieren Sie weiterhin die Methode *StrukturAusgeben* entsprechend den Erläuterungen in diesem Kapitel.

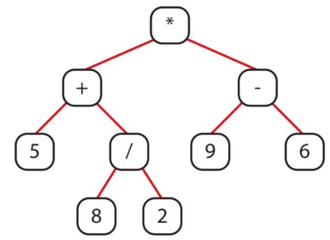
d Erläutern Sie, warum die beiden Binärbäume in der folgenden Abbildung beim Aufruf der Methode *StrukturAusgeben* auf der Konsole die gleiche Ausgabe erzeugen.



e Erstellen Sie eine Methode *StrukturAusgeben2*, bei der auch ABSCHLUSS-Objekte entsprechend der Hierarchie eine Ausgabe erzeugen, z. B. „a" für Abschluss.

5 Termbäume

a Notieren Sie das Ergebnis des Preorder- , Inorder- und Postorder-Durchlaufs für den nebenstehenden Termbaum.



b In allen Fällen von a) ist eine Ergänzung des Terms durch Klammern übersichtlicher. Ergänzen Sie passende Klammern. (Damit zwei aufeinander folgende Ziffern nicht als eine Zahl gelesen werden, sollten Sie einen deutlichen Abstand zwischen den Ziffern lassen oder Strichpunkte als Trennzeichen ergänzen.)

c Begründen Sie, warum in einem Fall ohne Klammern keine eindeutige Berechnung erfolgen kann.

d Für Schnelle: Begründen Sie, warum für die Berechnung des Termwerts ein Postorder-Durchlauf gemacht werden muss.

6 Nutzerverwaltung erweitern

a Ergänzen Sie in dem Projekt aus der Einstiegsaufgabe eine Methode *AnzahlKnotenBerechnen*, die die Anzahl der im Baum enthaltenen Knoten zurückgibt. Dokumentieren Sie Ihre Tests der Methode.

b Nennen Sie Gemeinsamkeiten und Unterschiede der Methode aus a) zu den Baumdurchläufen.

c Um einen Überblick über die Altersstruktur der Nutzenden zu erhalten, muss die Anzahl der Erwachsenen und der Jugendlichen bestimmt werden. Schreiben Sie dazu die Methode *AnzahlJugendlicheBerechnen* und *AnzahlErwachseneBerechnen*.

d Erweitern Sie das Projekt um eine weitere Funktionalität. Geben Sie eine Anwendungssituation für Ihre Ergänzung an.

2.5 Üben und Vertiefen: Weiterführende Aufgaben

Die folgenden Aufgaben haben als Ziel das Üben und Vertiefen der bisherigen Inhalte.

1 Burning Man (aus Abitur 2021 – Fortsetzung von S. 11)

Zur Speicherung aller auf dem Festival ausgestellten Kunstwerke wird ein geordneter Binärbaum verwendet, der lexikographisch nach dem eindeutigen Namen des Kunstwerks geordnet ist. Bei der Implementierung sollen das Konzept der Trennung von Struktur und Daten sowie das Entwurfsmuster Kompositum berücksichtigt werden.

a Der geordnete Binärbaum wird unter anderem zur Suche nach Kunstwerken mit einem bestimmten Namen bzw. nach Kunstwerken eines bestimmten Künstlers verwendet. Entscheiden Sie jeweils begründet, ob sich dabei ein Vorteil gegenüber einer ebenso nach dem Namen des Kunstwerks geordneten Liste ergeben kann.

b Der Ort eines Kunstwerks muss z. B. wegen schlechten Wetters kurzfristig innerhalb des Festivalgeländes geändert werden können.
Die Klasse KUNSTWERKBAUM soll zu diesem Zweck eine Methode *ortAendern(name, ortNeu)* besitzen, die dem Kunstwerk mit dem Namen name den Ort ortNeu zuweist.
Notieren Sie in Ihrer Programmiersprache eine Implementierung dieser sowie aller dafür in den Klassen der Baumstruktur notwendigen Methoden. Wenden Sie das Prinzip der Rekursion an.
Hinweise: Die Klasse KUNSTWERK darf einschließlich ihrer Standardmethoden zum Lesen und Setzen von Attributwerten als implementiert vorausgesetzt werden, ebenso eine Methode *vergleichenMit(zk)* der Klasse ZEICHENKETTE, die
- eine negative ganze Zahl zurückgibt, wenn das ausführende Objekt lexikographisch kleiner als die übergebene Zeichenkette zk ist,
- eine positive ganze Zahl zurückgibt, wenn das ausführende Objekt lexikographisch größer als die übergebene Zeichenkette zk ist,
- die Zahl 0 zurückgibt, wenn die Zeichenketten übereinstimmen.

Zur Datensicherung wird der geordnete Binärbaum mittels Preorder-Traversierung durchlaufen. Die ausgelesenen Daten werden dabei der Reihe nach in eine Datei gespeichert.

c Folgende Kunstwerke wurden nacheinander ausgelesen:
„DesertWave“, „Carousel“, „Bee“, „Chakra“, „GiantPinball“, „Fascination“, „Paraluna“
Zeichnen Sie den zugrunde liegenden Baum.

d Begründen Sie kurz, weshalb sich die Preorder-Traversierung zur Sicherung von Daten, die in einem geordneten Binärbaum gespeichert sind, als vorteilhaft gegenüber anderen Traversierungen erweist.

2 Schulverwaltungssoftware (aus Abitur 2022 – Fortsetzung von S. 10)

Die Schülerinnen und Schüler des Wahlkurses „Homepage“ möchten zum neuen Schuljahr einen Schulnewsletter einrichten, um z. B. über Veranstaltungen der Schule zu informieren. Interessierte können den Newsletter abonnieren. Für die Verwaltung der Abonnentinnen und Abonnenten wird ein lexikographisch geordneter Binärbaum verwendet, wobei eine eindeutige E-Mail-Adresse als Schlüssel dient.

a Vor Einführung des Schulnewsletters testen die Schülerinnen und Schüler ihre Implementierung des Baums mithilfe von Beispieldaten. Ein Testdurchlauf für alle im Baum enthaltenen E-Mail-Adressen ergibt folgende Ausgabereihenfolge:

georg@beispiel.de
dana@example.com
paul@example.net
maja@beispiel.de
werner@example.org
silke@infohausen.de
karl@example.edu

Begründen Sie, dass die gegebene Ausgabe weder durch Inorder-Durchlauf noch durch Preorder-Durchlauf erzeugt worden sein kann.
Zeichnen Sie unter der Annahme, dass die Ausgabe durch einen Postorder-Durchlauf erzeugt wurde, den dazugehörigen Baum. Dabei können Sie sich bei den Bezeichnern der Knoten auf die gegebenen Vornamen beschränken.

b Geben Sie eine Einfügereihenfolge der in Teilaufgabe a) genannten E-Mail-Adressen an, sodass der Binärbaum möglichst wenige Ebenen besitzt. Dabei genügt es, wieder nur die Vornamen anzugeben. Der Baum muss nicht erneut gezeichnet werden.

c Der Schulnewsletter wird von 2000 Personen abonniert. Geben Sie begründet die maximal sowie die minimal nötige Anzahl der Ebenen für einen dazugehörigen Binärbaum an.

d Der Binärbaum wurde unter Verwendung der Klassen BAUM, BAUMELEMENT, KNOTEN, ABSCHLUSS und ABONNENT sowie des Entwurfsmusters Kompositum unter Berücksichtigung der Trennung von Struktur und Daten umgesetzt. Die Klasse ABONNENT wird durch nebenstehendes Klassendiagramm beschrieben:

ABONNENT
eMail name
istKleinerAls(text) istGroesserAls(text) ausgeben()

Für die Methoden der Klasse ABONNENT gilt:
- *istKleinerAls(text)* gibt genau dann wahr zurück, wenn der Wert des Attributs eMail des ausführenden Abonnent-Objekts lexikographisch kleiner als der Wert des Parameters text ist, wobei zwischen Groß- und Kleinschreibung nicht unterschieden wird.
- *istGroesserAls(text)* gibt genau dann wahr zurück, wenn der Wert des Attributs eMail des ausführenden Abonnent-Objekts lexikographisch größer als der Wert des Parameters text ist, wobei zwischen Groß- und Kleinschreibung nicht unterschieden wird.
- *ausgeben()* gibt die E-Mail-Adresse auf dem Bildschirm aus.

In der Klasse BAUM soll nun eine Methode *eMailsAusgeben(von, bis)* programmiert werden, die alle E-Mail-Adressen in alphabetischer Reihenfolge ausgibt, die lexikographisch zwischen den beiden übergebenen Zeichenketten von und bis eingeordnet sind, wobei die Grenzen von bzw. bis nicht mit ausgegeben werden. Z. B. soll der Methodenaufruf *eMailsAusgeben("dana", "karl")* für den oben beschriebenen Baum die E-Mail-Adressen dana@example.com und georg@beispiel.de ausgeben, da dana@example.com lexikographisch größer als dana ist.
Notieren Sie […] eine mögliche Implementierung der Methode *eMailsAusgeben(von, bis)* in der Klasse BAUM und aller dazu nötigen Methoden in den weiteren Klassen der Baumstruktur. Verwenden Sie so weit wie möglich das Prinzip der Rekursion und nutzen Sie die Vorteile des geordneten Binärbaums. Die Klasse ABONNENT kann als bereits implementiert vorausgesetzt werden.

3 Adressbuch – Binärbaumstruktur in anderen Anwendungen einsetzen

Die bisher für das Wörterbuch verwendete Baumstruktur aus den Lehrtexten bzw. Aufgaben soll für ein Adressbuch verwendet werden. Sollten Sie sich im Programmieren sicher fühlen, dann setzen Sie ein Adressbuch als Programm um. Testen und dokumentieren Sie das Programm ausführlich.

Falls Sie Hilfestellungen benötigen, bearbeiten Sie schrittweise folgende Teilaufgaben.

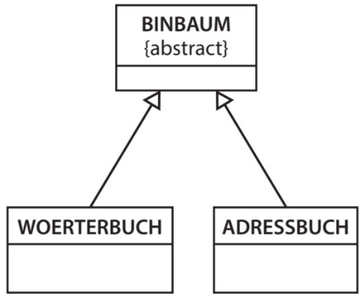
a Notieren Sie die gewünschten Funktionalitäten eines Adressbuchs.

b Notieren Sie, an welchen Stellen das Klassendiagramm aus Aufgabe 3 des Kapitels 2.3 (S. 67) verändert werden muss, um diese Funktionalitäten zu ermöglichen.
Zeichnen Sie insbesondere eine Klassenkarte von der Klasse ADRESSBUCHEINTRAG.
(Kontrollmöglichkeit am Ende dieses Kapitels)

c Führen Sie nun in einer Kopie von Ihrem Wörterbuch-Projekt schrittweise die Änderungen aus b) durch. Beginnen Sie dabei mit der Klasse ADRESSBUCHEINTRAG.
Sollten Sie Kapitel 1.3 bearbeitet haben, lesen Sie die Hinweise in d) und e).

***d** Ist die in Kapitel 1.3 beschriebene Schnittstelle DATENELEMENT auch in der Binärbaum-Struktur vorhanden, ist die Umsetzung für andere Anwendungen leichter: Zusätzlich zum WÖRTERBUCHEINTRAG kann auch der ADRESSBUCHEINTRAG die Schnittstelle implementieren. Dadurch können alle bisherigen Funktionalitäten des Binärbaums genutzt werden.

***e** Eine vom Modell gesehen schönere, aber von der Struktur her komplexere Lösung für die Teilaufgabe c) besteht darin, nicht die Klasse BINBAUM für Adressbuchfunktionalitäten umzuschreiben, sondern das Adressbuch als eine Spezialisierung des Binärbaums zu sehen. Entsprechend wäre auch das Wörterbuch eine Spezialisierung. Die Klasse BINBAUM wäre somit eine abstrakte Oberklasse, die dann durch konkrete Anwendungen ausgeführt wird.



In den Spezialisierungen müssen Typüberprüfungen zu den passenden Unterklassen der Klassen DATENELEMENT, WOERTERBUCHEINTRAG und ADRESSBUCHEINTRAG durchgeführt werden.

Weiterhin können entsprechend der Anforderungen der konkreten Anwendung Methoden ergänzt bzw. nicht benötigte Methoden der Klasse BINBAUM mit einem leeren Methodenrumpf überschrieben werden.

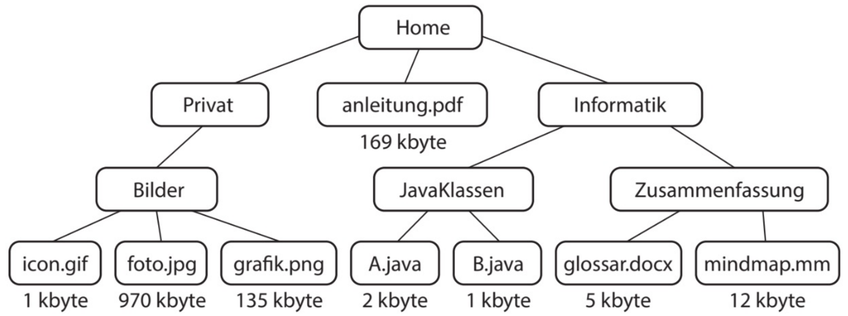
Implementieren Sie eine solche optimierte Lösung, die dann sehr schnelle Umsetzungen weiterer Anwendungen der Datenstruktur Binärbaum ermöglicht.

***4 Verzeichnisbaum mit Dateien**

Ein Dateisystem lässt sich einfach als Baum darstellen (siehe Abbildung auf S. 77). Die Blätter repräsentieren die Dateien. Die inneren Knoten entsprechen den Verzeichnissen oder Ordnern. Die Grafik (siehe Abbildung auf S. 77) zeigt einen Ausschnitt aus einem Dateisystem mit Angabe der Dateigrößen. Die Verzeichnisse selbst benötigen dabei keinen Speicherplatz. Ziel ist es, den benötigten Speicherplatz der in dem Baum gespeicherten Dateien zu berechnen.

a Zeichnen Sie ein zu dieser Problemstellung passendes Klassendiagramm (mit Beziehungen, jedoch ohne Attribute und Methoden).

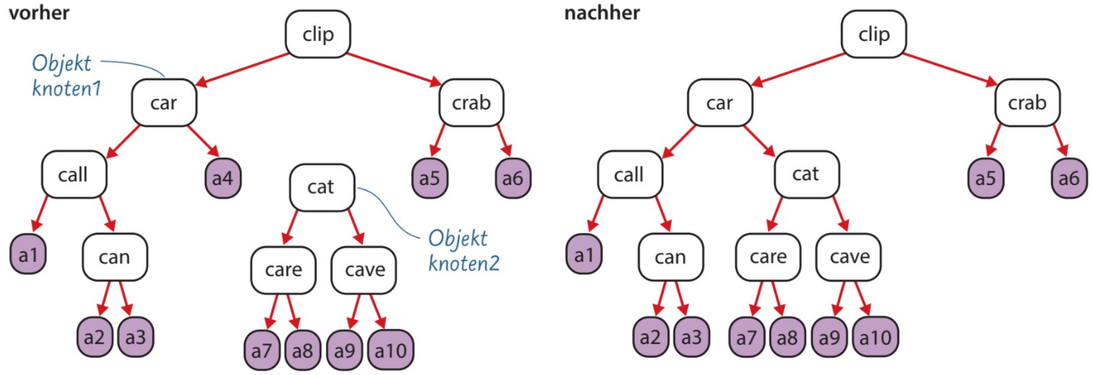
b Skizzieren Sie stichwortartig, wie der Speicherplatz der in dem Baum gespeicherten Dateien berechnet werden kann. Gehen Sie dabei darauf ein, in welchen Klassen aus a) welche Methoden bereitgestellt werden müssen.



***5 Weitere rekursive Methoden eines Binärbaums**

Erweitern Sie die Klassen des Binärbaums zum Wörterbuch um die beiden Methoden *Rechts-Einfügen* und *Entfernen*. Achten Sie auf eine rekursive Formulierung, die den Vorteil des Entwurfsmusters Kompositum ausnutzt. Gehen Sie schrittweise entsprechend den folgenden Teilaufgaben vor.

a Die Methode *RechtsEinfügen(element: BAUMELEMENT)* fügt ein Baumelement „rechts unten" in den Teilbaum ein, dessen Wurzel das aufgerufene Objekt ist. Die folgende Abbildung zeigt die Situation vor und nach dem Aufruf der Methode *knoten1.RechtsEinfügen(knoten2)*.



Hinweis: Da das einzufügende Baumelement Nachfolger haben kann und in diesem Fall Wurzel eines Teilbaumes ist, können mit dieser Methode verschiedene Teilbäume zusammengefügt werden. Überlegen Sie sich, welche Anforderungen gelten müssen, damit die Ordnung des Binärbaums nicht verletzt wird.

Eine wichtige, aber komplexe Methode einer Baumstruktur ist *Entfernen(suchschlüssel: ZEICHENKETTE)*. Sie entfernt einen Knoten, dessen Datenelement den eingegebenen Schlüssel hat, aus dem Teilbaum, dessen Wurzel der aufgerufene Knoten ist. Dabei bleibt die Struktur „geordneter Binärbaum" erhalten.

b Man muss drei unterschiedliche Fälle unterscheiden. Experimentieren Sie mit der Methode *Entfernen* im BaumVisualisierungsTool, indem Sie diese mit unterschiedlichen Eingabewerten / Bäumen ausführen.
Beschreiben Sie die unterschiedlichen Fälle und formulieren Sie jeweils informell als Text den Algorithmus zum Entfernen eines Knotens.
Hinweis: Am Ende dieses Kapitels steht Ihnen zur Überprüfung Ihrer Lösung dieser Teilaufgabe eine Lösungsskizze zur Verfügung.

c Ergänzen Sie in Ihrem Projekt in den Klassen BAUMELEMENT, KNOTEN, ABSCHLUSS und BINBAUM eine Methode Entfernen. Gehen Sie schrittweise wie in b) erarbeitet vor, indem Sie jeden Fall beim Entfernen nacheinander implementieren und die Lösung ausführlich testen.

d Überprüfen Sie auf Papier an drei Beispielen, die die unterschiedlichen Fälle aus b) repräsentieren, dass auch folgende Strategie zum Entfernen genutzt werden kann: Beim Knoten, dessen Datenelement den suchschlüssel enthält, wird zunächst folgende Methode aufgerufen: *linkerNachfolger.RechtsEinfügen(rechterNachfolger)*;
Der Rückgabewert ersetzt dann den Knoten und entfernt ihn dadurch.

e Implementieren Sie die Methode *Entfernen2(suchschlüssel)*, die die Strategie aus d) umsetzt.

6 Buchstabenhäufigkeit in Texten bestimmen

Für einige Verschlüsselungs- und Komprimierungsalgorithmen ist es wichtig, die Häufigkeiten der Buchstaben in einem Text zu kennen. Ziel dieser Aufgabe ist es ein Programm zu entwickeln, das mit Hilfe eines geordneten Binärbaums für einen beliebigen Text die absolute Häufigkeit der Buchstaben bestimmen und ausgeben kann.

a Implementieren Sie zunächst eine Klasse TEXTZEICHEN als Datenelement (bzw. als Implementierung der Schnittstelle DATENELEMENT, wenn Sie Kapitel 1.3 behandelt haben). Die Klasse hat zwei Attribute: textzeichen und die anzahl, wie oft das Zeichen bereits vorkommt. Die Anzahl soll beim Erzeugen eines Buchstaben-Objekts auf 1 initialisiert werden, da es das erste Vorkommen dieses Zeichens ist. Testen Sie alle Methoden und dokumentieren Sie die Tests – es ist eine wichtige Grundlage für den korrekten Ablauf anderer Methoden.

TEXTZEICHEN
textzeichen: ZEICHEN anzahl: GANZZAHL
Konstruktor(ZEICHENKETTE, ZEICHENKETTE) AnzahlErhöhen() AnzahlGeben(): GANZZAHL TextzeichenGeben(): ZEICHEN InformationAusgeben() IstGleich(DATENELEMENT): WAHRHEITSWERT IstGrößerAls(DATENELEMENT): WAHRHEITSWERT SchlüsselIstGleich(ZEICHENKETTE): WAHRHEITSWERT SchlüsselIstGrößerAls(ZEICHENKETTE): WAHRHEITSWERT SchlüsselGeben(): ZEICHENKETTE

b Über das schrittweise Einfügen von den Textzeichen in den Binärbaum soll die absolute Häufigkeit der Buchstaben in den Datenelementen gezählt werden. Erstellen Sie dazu die Methode *ZeichenEinfügenUndZählen*, um diese Funktionalität umzusetzen.
Hinweise:
- Orientieren Sie sich an der Methode *Einfügen* und erweitern Sie diese passend.
- Das Vorgehen verletzt die Trennung von Struktur und Inhalt. Im Rahmen einer schnellen Problemlösung ist die Adaption der Datenstruktur an die Anforderungen jedoch akzeptiert.

c Erstellen Sie eine Klasse HÄUFIGKEITSANALYSE, die den Binärbaum referenziert. Ein Aufruf der Methode *TextAnalysieren(String text)* soll den Text zeichenweise einlesen und danach die Häufigkeitsanalyse auf der Konsole ausgeben.
Zum Testen erhalten Sie sowohl einen längeren englischen als auch längeren deutschen Text.

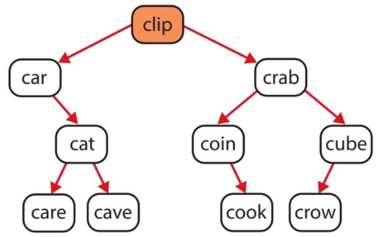
d Für die Identifikation der Sprache sind Groß-/Kleinschreibung und Satzzeichen nicht relevant. Implementieren Sie eine Methode *TextAnalysieren2(String text)*, die Satz- und Leerzeichen ignoriert sowie nicht zwischen Groß- und Kleinschreibung unterscheidet.

e Für Schnelle: Erweitern Sie das Programm so, dass eine Textdatei eingelesen werden kann und dann die Analyse mit der eingelesenen Textdatei durchgeführt wird.

Kontrollergebnis zu Aufgabe 3b
Siehe Klassenkarte rechts

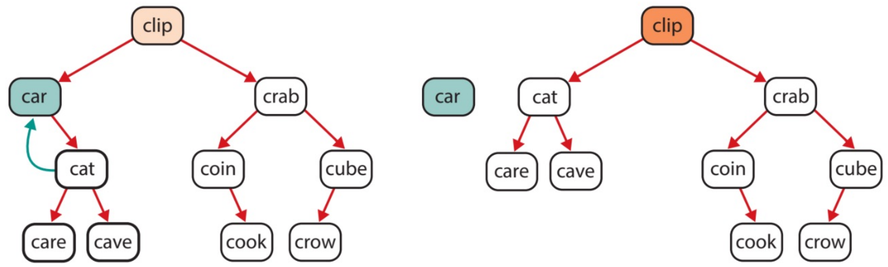
ADRESSBUCHEINTRAG
name: ZEICHENKETTE telefonnummer: ZEICHENKETTE e-mail: ZEICHENKETTE
Konstruktor(ZEICHENKETTE, ZEICHENKETTE) Konstruktor(ZEICHENKETTE, ZEICHENKETTE, ZEICHENKETTE) InformationAusgeben() IstGleich(DATENELEMENT): WAHRHEITSWERT IstGrößerAls(DATENELEMENT): WAHRHEITSWERT SchlüsselIstGleich(ZEICHENKETTE): WAHRHEITSWERT SchlüsselIstGrößerAls(ZEICHENKETTE): WAHRHEITSWERT SchlüsselGeben(): ZEICHENKETTE NameGeben(): ZEICHENKETTE

Kontrollergebnis zu Aufgabe 5b
Fall 1 – der zu entfernende Knoten ist ein Blatt
Vorgehensweise beim Entfernen: Der Knoten wird durch einen Abschluss ersetzt.
Fall 2 – der zu entfernende Knoten hat einen Nachfolger, z. B. „car" in folgendem Beispielbaum:



Vorgehensweise beim Entfernen: Es kann der unter dem zu löschenden Knoten liegende Teilbaum „nach oben geschoben" werden.
Das Schieben des Teilbaums nach oben bedeutet im Beispiel, dass dem Referenzattribut linkerNachbar des Wurzelknotens eine Referenz auf den Knoten mit dem Datenelement mit dem Schlüssel `"cat"` zugewiesen werden muss. Dies lässt sich am einfachsten dadurch erreichen, dass die Methode *Entfernen* eine Referenz auf ein Baumelement als Rückgabewert hat. Im Beispiel oben bedeutet dies:

- Der Rückgabewert des Aufrufs wurzel.linkerNachbar.Entfernen(`"car"`) ist eine Referenz auf den Knoten mit dem Datenelement `"cat"`.
- Diese Referenz wird dem Attribut wurzel.linkerNachbar zugewiesen.



Fall 3 – der zu entfernende Knoten hat zwei Nachfolger, z. B. „crab" im Beispielbaum oben.

Vorgehensweise beim Entfernen:
Man ersetzt den Knoten durch den Knoten mit dem kleinsten Schlüssel aus dem Teilbaum, der vom rechten Nachfolger des zu entfernenden Knotens ausgeht. Beispielsweise würde in der Abbildung oben beim Methodenaufruf wurzel.Entfernen(`"crab"`) der Knoten mit dem Schlüssel `"crab"` durch den Knoten mit dem Schlüssel `"crow"` ersetzt. Der Binärbaum bleibt bei dieser Vorgehensweise geordnet.

Teste dich selbst

T1 Rekursive Datenstruktur Baum und Entwurfsmuster Kompositum

a Beschreiben Sie anhand eines Beispiels Vorteile, die die Datenstruktur Baum im Vergleich zur Datenstruktur Liste haben kann.

b Nehmen Sie Stellung zur Aussage „Ein Binärbaum mit 500 Elementen hat maximal neun Ebenen.".

Alle folgenden Teilaufgaben beziehen sich auf eine Implementierung mit Entwurfsmuster Kompositum.

c Zeichnen Sie das Klassendiagramm für die Datenstruktur Binärbaum. Auf Attribute und Methoden können Sie verzichten, Schwerpunkt ist die Struktur.
Erläutern Sie, warum ein Baum eine rekursive Datenstruktur ist.

d Begründen Sie, warum aus dem Klassendiagramm nicht ersichtlich ist, ob der Binärbaum geordnet ist.

e Erklären Sie anhand von c), an welchen Stellen Polymorphismus zur Umsetzung verwendet wird.

f In einen leeren geordneten Binärbaum zur Verwaltung von Kunden wurde ein Kunde erfolgreich eingefügt. Zeichnen Sie die Objektdiagramme des Baums vor und nach dem Einfügen (mit den Objekten aller beteiligten Klassen). Sie müssen keine Attribute und Attributwerte angeben, aber aussagekräftige Objektnamen verwenden.

g Erklären Sie anhand von f) die Arbeitsweise der Methode *Einfügen*.

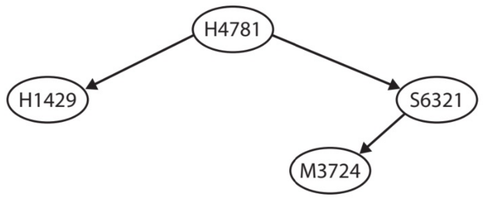
T2 Ich check's, dank deiner Hilfe!

Ferdi hat im Unterricht nicht gut aufgepasst und braucht dringend Ihre Nachhilfe vor der alles entscheidenden letzten Prüfung. Helfen Sie ihm, indem Sie ihm die folgenden Begriffe an Beispielen erklären, und zeigen, wie man damit umgeht:
vollständiger, entarteter sowie ausbalancierter Baum; Knoten; Wurzel; Kante; Blatt; Pfad; Höhe; Ebene; geordneter Binärbaum; rekursive Methoden zum Einfügen, Suchen und Traversieren.

T3 Kundenverwaltung Spieleverleih (aus Abitur 2020)

Die Kundendaten sollen in einem geordneten Binärbaum verwaltet werden. Die Daten werden lexikographisch nach den eindeutigen Kundennummern sortiert. Die Kundennummer wird aus dem Anfangsbuchstaben des Nachnamens sowie einer vierstelligen positiven Zahl gebildet.

a Gegeben ist folgender geordneter Binärbaum:



Geben Sie jede mögliche Reihenfolge an, in der die Kundennummern in einen anfangs leeren Baum eingefügt worden sein können, sodass der gegebene Baum entsteht. Zeichnen Sie außerdem den Baum, der entsteht, wenn in den gegebenen Baum nun nacheinander die Kundennummern R7491, N2855 sowie L7762 eingefügt werden.

b Nennen und beschreiben Sie ein Verfahren, um alle Kunden nach Kundennummern sortiert auszugeben. Begründen Sie kurz, warum dieses Verfahren in der Regel keine nach den Nachnamen der Kunden sortierte Ausgabe erzeugt.

c Der Spieleverleih rechnet in der Anfangsphase mit 2500 Kunden. Bestimmen Sie die Anzahl der Ebenen, die ein dazugehöriger geordneter Binärbaum mindestens haben muss.

d Nennen Sie je einen Vor- und einen Nachteil der Implementierung von Maßnahmen, die sicherstellen, dass die Ebenenanzahl des Binärbaums stets minimal ist.

Bei der Implementierung des geordneten Binärbaums zur Verwaltung der Kundendaten mithilfe einer Klasse KUNDENBAUM kommen das Entwurfsmuster Kompositum sowie das Konzept der Trennung von Struktur und Daten zum Einsatz.

e Zeichnen Sie das zugehörige Klassendiagramm. Auf Attribute und Methoden können Sie verzichten, Schwerpunkt ist die Struktur. (*nicht Teil der Abituraufgabe*)

f Die Klasse KUNDE ist folgendermaßen beschrieben:

KUNDE
kundenummer nachname vorname passwort
kundenummerVergleichen(kdnr) istPasswortGleich(pw)

Die Methode *kundennummerVergleichen(kdnr)*

- gibt 1 zurück, wenn der Parameter kdnr einen größeren Wert als das Attribut kundennummer des ausführenden Objekts hat,
- gibt 0 zurück, wenn der Parameter kdnr den gleichen Wert wie das Attribut kundennummer des ausführenden Objekts hat,
- gibt −1 zurück, wenn der Parameter kdnr einen kleineren Wert als das Attribut kundennummer des ausführenden Objekts hat.

Die Methode *istPasswortGleich(pw)* gibt genau dann wahr zurück, wenn der Wert des Parameters pw mit dem des Attributs passwort des ausführenden Objekts übereinstimmt.

Um sich online beim Spieleverleih anzumelden, muss ein Kunde seine Kundennummer und sein Passwort eingeben. Die Klasse KUNDENBAUM soll eine Methode *passwortPruefen (kdnr, pw)* bereitstellen, die genau dann wahr zurückgibt, wenn in der Baumstruktur ein Kunde mit Kundennummer kdnr und Passwort pw eingetragen ist.

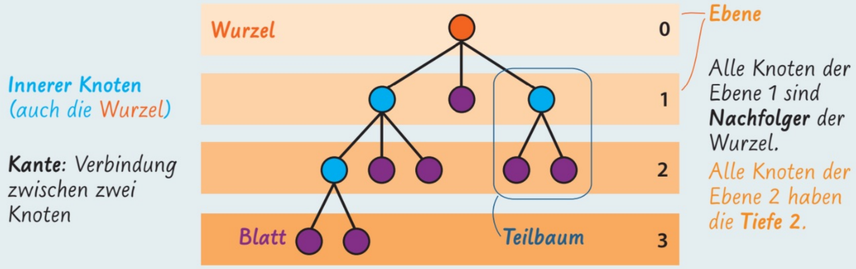
Notieren Sie […] eine mögliche Implementierung dieser Methode für die Klasse KUNDENBAUM und der dafür nötigen Methoden aller weiteren Klassen der Baumstruktur. Nutzen Sie dabei das Ordnungsprinzip des Binärbaums. Die Klasse KUNDE kann als vollständig implementiert vorausgesetzt werden.

Zusammenfassung

Datenstruktur Baum: Bestandteile und Eigenschaften

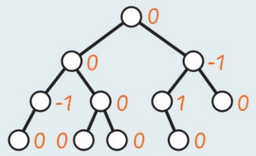
In der Datenstruktur **Baum** hat jeder Knoten außer der Wurzel genau einen Vorgänger. Die **Tiefe** eines Knotens ist ein Maß für die Länge des Pfades bis zur Wurzel. **Höhe** eines Baumes ist die Tiefe des Knotens, der am weitesten von der Wurzel entfernt ist. Als einen **Pfad** zwischen zwei Knoten A und B bezeichnet man die Folge der (verschiedenen) Knoten, die aufeinanderfolgend durch eine Kante miteinander verbunden sind.
Ein **Binärbaum** ist ein Baum, bei dem jeder Knoten höchstens zwei Nachfolger hat. In einem Binärbaum mit n Ebenen können **maximal $2^n - 1$** Knoten gespeichert werden.

Beispiel 1: unbalancierter Baum mit der Höhe 3, der kein Binärbaum ist



Ein **geordneter Binärbaum** ist ein Binärbaum, bei dem im linken Teilbaum jedes Knotens nur kleinere und im rechten Teilbaum nur größere Datenelemente bezüglich einer Ordnungsrelation stehen (oder umgekehrt).

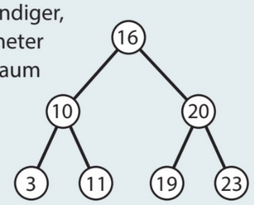
Beispiel 2: balancierter Binärbaum



Höhendifferenz = Höhe (rechter Teilbaum) - Höhe (linker Teilbaum)

balanciert: Höhendifferenz ist 0, 1 oder -1

Beispiel 3: vollständiger, geordneter Binärbaum



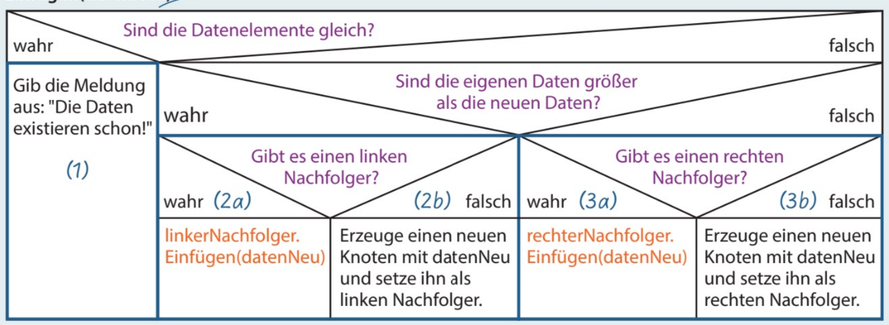
vollständig: passend zur Höhe maximale Knotenanzahl erreicht

Rekursive Algorithmen im geordneten Binärbaum: Einfügen, Suchen, Traversierungen

Wichtige Methoden der Datenstruktur geordneter Binärbaum sind das **Suchen** und das **Einfügen** von Datenelementen. Beide Methoden arbeiten rekursiv mit mehreren Abbruchbedingungen und rekursiven Aufrufen.

Variante ohne Entwurfsmuster Kompositum

Einfügen(datenNeu)



Sowohl über den Preorder-, den Inorder- als auch den Postorder-Durchlauf werden alle Knoten eines Binärbaums erreicht. Die Durchlaufarten (Traversierungen) unterscheiden sich darin, ob der eigene Inhalt vor, zwischen oder nach den rekursiven Aufrufen an den linken und rechten Nachfolger ausgegeben werden.
Typische Anwendungen für den Preorder-Durchlauf sind eine Ausgabe der hierarchischen Struktur bzw. das Abspeichern eines Baumes. Der Inorder-Durchlauf wird eingesetzt, um die Daten sortiert auszugeben.

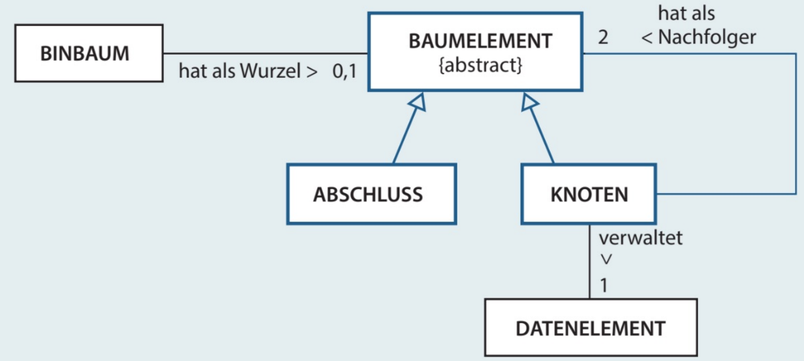
Variante mit Entwurfsmuster Kompositum

Klasse KNOTEN
Methode PreorderAusgeben()

```
daten.InformationAusgeben()
linkerNf.PreorderAusgeben()
rechterNf.PreorderAusgeben()
```

Implementierung des Binärbaums

Bei der Implementierung eines geordneten Binärbaums ist die Trennung von Struktur und Inhalt wichtig. Da über Nachfolgerbeziehungen die Ordnung umgesetzt wird, sind in den Methoden Einfügen und Suchen entsprechende Fallunterscheidungen nötig. Der Einsatz des Entwurfsmusters Kompositum sorgt für kompakte Methodenrümpfe.



Klasse LISTE
Methode Suchen(suchschlüssel:ZEICHENKETTE)->DATENELEMENT

```
return anfang.Suchen(suchschlüssel)
```

Klasse KNOTEN beerbt LISTENELEMENT
Methode Suchen(suchschlüssel:ZEICHENKETTE)->DATENELEMENT

```
falls daten.SchlüsselIstGleich(suchschlüssel)
dann
  return daten
sonst
  falls daten.IstSchlüsselGrößerAls(suchschlüssel)
  dann
    return linkerNachfolger.Suchen(suchschlüssel)
  sonst
    return rechterNachfolger.Suchen(suchschlüssel)
  endefalls
endefalls
```

Klasse ABSCHLUSS beerbt LISTENELEMENT
Methode Suchen(suchschlüssel:ZEICHENKETTE)->DATENELEMENT

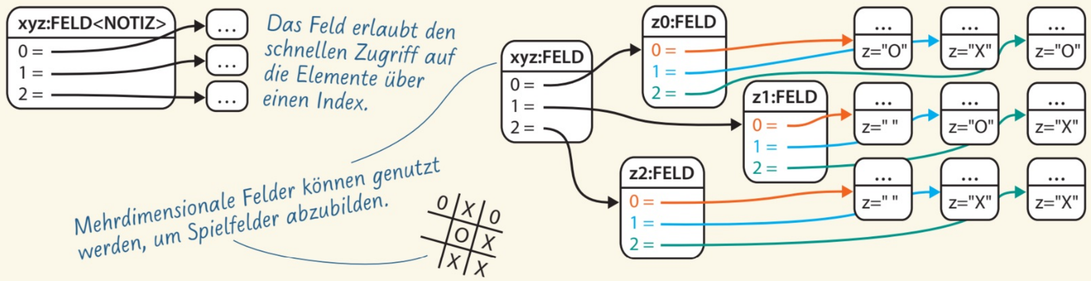
```
return null
```

Der Aufruf der rekursiven Methode bei einem Abschluss-Objekt hat das Rekursionsende zur Folge.

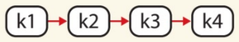
Zum Weiterlesen

L2 Übersicht Datenstrukturen

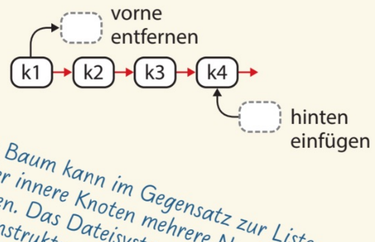
Die Informatik bietet neben dem Baum eine Fülle an weiteren Datenstrukturen für verschiedene Einsatzzwecke, von denen Sie einige bereits kennengelernt haben:



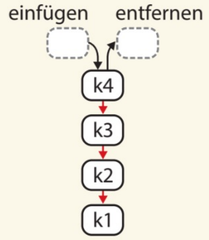
Die Liste ermöglicht den rekursiven Zugriff auf die Elemente. Spezielle Listen sind...

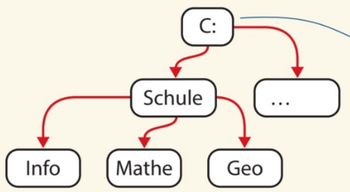


...Warteschlange als FIFO-Speicher (First in, first out) und ...



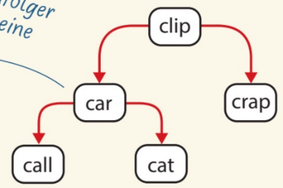
...Stapel als LIFO-Speicher (Last in, first out)





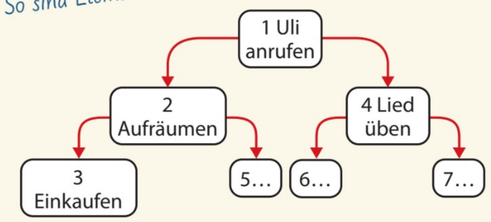
Im Baum kann im Gegensatz zur Liste jeder innere Knoten mehrere Nachfolger haben. Das Dateisystem hat z. B. eine Baumstruktur.

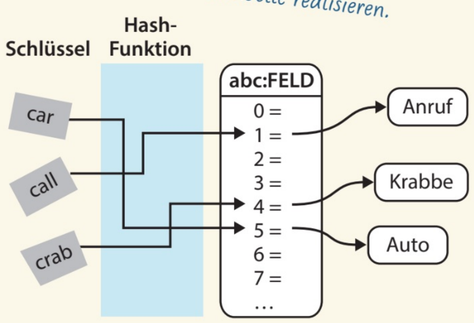
Im geordneten Binärbaum sind links von einem Knoten alle Elemente kleiner, rechts davon alle Elemente größer (oder umgekehrt). In dieser Datenstruktur ist das Suchen geordneter Daten (z. B. Wörterbuch) besonders effizient.



Ein Wörterbuch ließe sich auch effizient über eine Hashfunktion und ein Feld in einer sogenannten Hashtabelle realisieren.

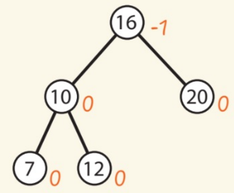
Ein Heap ist auch ein Binärbaum. Er ist aber nach Priorität von oben nach unten geordnet. Die wichtigsten Elemente stehen oben. Neue Elemente werden einfach unten eingefügt. Haben diese eine höhere Priorität, so wird getauscht. So sind Elemente mit höchster Priorität immer oben.





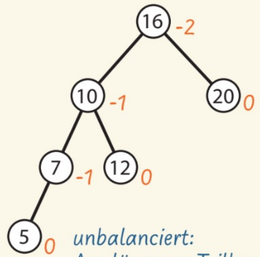
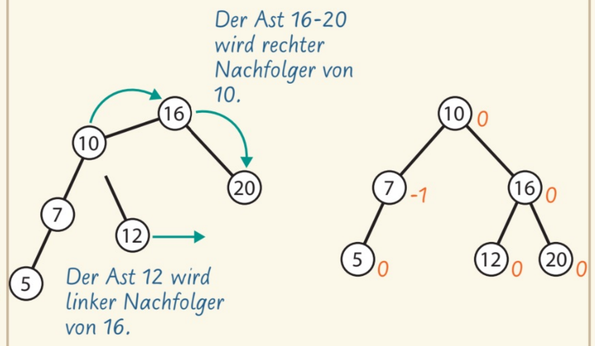
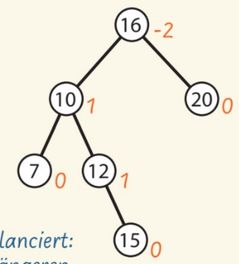
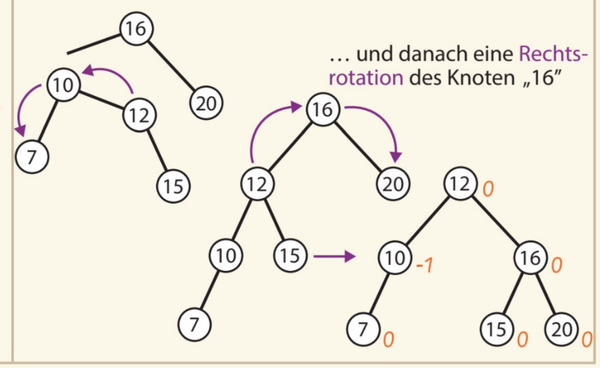
L3 AVL-Bäume

Durch ungünstige Situationen beim Löschen oder Einfügen können die Äste eines Baumes sehr unterschiedliche Längen erreichen. Wenn sich die Höhe im linken und rechten Teilbaum eines Knotens stark unterscheidet, ist die Suche jedoch nicht effizient. Deshalb sind balancierte Bäume erwünscht, bei denen die Höhendifferenz in jedem Knoten nur 0, 1 oder -1 beträgt (siehe Beispiel rechts). Dieses Kriterium wurde von den russischen Informatikern G. M. **A**delson-**V**elskii und J. M. **L**andis aufgestellt; deshalb nennt man diese Bäume auch AVL-Bäume.



Höhendifferenz = Höhe (rechter Teilbaum) - Höhe (linker Teilbaum)

Damit der AVL-Baum auch nach einem Einfügen höhenbalanciert ist, müssen ggf. Höhenunterschiede ausgeglichen werden:

Nach dem „normalen“ Einfügen werden die Höhen an den Knoten neu berechnet.	Hat sich die Höhendifferenz an Knoten auf -2 oder +2 verändert, wird dies durch sogenannte Rotationen korrigiert.
Beispiel 1: Einfügen von „5“  *unbalanciert: Am längeren Teilbaum ist der Außenast länger.*	im Beispiel 1: Rechtsrotation des Knoten „16“  *Der Ast 16-20 wird rechter Nachfolger von 10.* *Der Ast 12 wird linker Nachfolger von 16.*
Beispiel 2: Einfügen von „15“  *unbalanciert: Am längeren Teilbaum ist der Innenast länger.*	im Beispiel 2: Linksrotation des Knoten „10“ …  … und danach eine Rechtsrotation des Knoten „16“

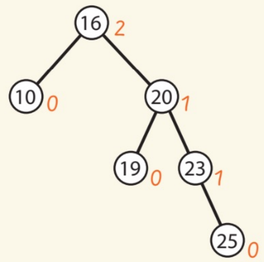
Das Rotieren lässt sich umsetzen durch ein Umspeichern der Referenzen:

Klasse KNOTEN
Methode Rechtsrotieren() -> BAUMELEMENT

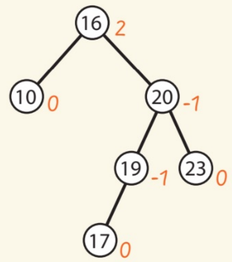
```
attribut ergebnis: KNOTEN
ergebnis = linkerNachfolger
linkerNachfolger = ergebnis.rechterNachfolger
ergebnis.rechterNachfolger = selbst
return selbst
```

Neben den oben ausführlich dargestellten Fällen mit längerem linken Teilbaum gibt es noch zwei dazu symmetrische Fälle mit einem längeren rechten Teilbaum. Überzeugen Sie sich (mit Stift und Papier), dass eine Linksrotation bzw. eine Rechtsrotation mit anschließender Linksrotation zu einem balancierten Baum führt.

Am längeren (rechten) Teilbaum ist der Außenast länger.



Am längeren (rechten) Teilbaum ist der Innenast länger.



Entsprechend den Vorüberlegungen ergibt sich folgender Algorithmus mit vier Fällen für die Methode *Ausgleichen* der Klasse KNOTEN:

Ausgleichen() -> BAUMELEMENT

```
attribut ergebnis: BAUMELEMENT
ergebnis = selbst
liNf = liNf.Ausgleichen()
reNf = reNf.Ausgleichen()
```

Rekursive Aufrufe, um bei allen Knoten zu überprüfen, ob ausgeglichen werden muss.

```
selbst.HöhendifferenzBerechnen() == -2
  wahr:
    liNf.HöhendifferenzBerechnen() == -1
      wahr:
        ergebnis = RechtsRotieren()
      falsch:
        liNf = liNf.LinksRotieren()
        ergebnis = RechtsRotieren()
    ergebnis = ergebnis.Ausgleichen()
  f:
selbst.HöhendifferenzBerechnen() == 2
  wahr:
    reNf.HöhendifferenzBerechnen() == -1
      wahr:
        reNf = reNf.RechtsRotieren()
        ergebnis = selbst.LinksRotieren()
      falsch:
    ergebnis = ergebnis.Ausgleichen()
  f:
return ergebnis
```

Nur bei einem Knoten mit einer Höhendifferenz von -2 bzw. 2 finden Rotationen statt.

Hat ein Knoten die Höhendifferenz 2 bzw. -2, so muss einer der Nachfolger die Höhendifferenz -1 bzw. Höhendifferenz 1 haben. Über die Höhendifferenz des Nachfolgers kann bestimmt werden, ob der Innen- oder Außenast länger ist.

Eine Rückgabe vom Typ BAUMELEMENT ist bei der Implementierung mit Entwurfsmuster Kompositum nötig, damit wie bei der Methode Einfügen Umstrukturierungen möglich sind.

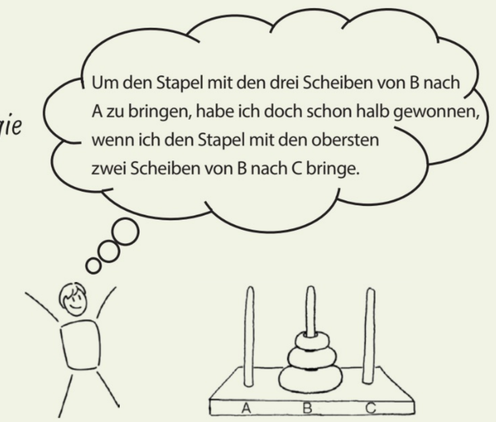
Beachte: Bei einem längeren linken Teilbaum bedeutet die -1 als Höhendifferenz des Nachfolgers einen längeren Außenast, bei einem längeren rechten Teilbaum dagegen einen längeren Innenast.

3 Rekursion

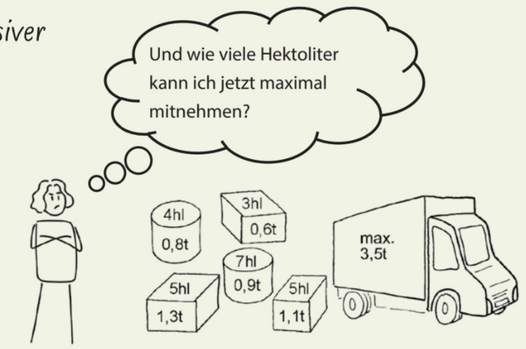
In diesem Kapitel werden Sie ...

... rekursives Vorgehen näher betrachten.

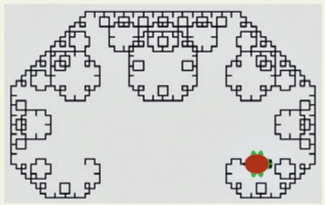
... Rekursion als Problemlösestrategie anwenden.



... typische Anwendungen rekursiver Vorgehensweise kennenlernen.



3.1 Methoden rufen sich selbst auf: Rekursion

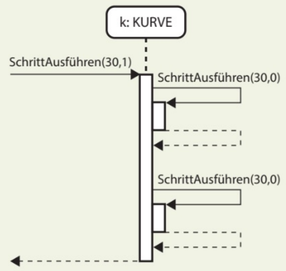


Nebenstehende Figur sieht sehr kompliziert aus, entsteht aber durch ein einfaches Programm.

a Erzeugen Sie ein Objekt der Klasse KURVE und rufen Sie die Methode *KurveZeichnen* mit den Werten 0 bis 5 und 10 für den Parameter tiefe sowie schwarz für die Farbe auf. Beschreiben Sie grob, wie sich die gezeichnete Kurve verändert.

b Rufen Sie nun die Methode *KurveZeichnen0bis3* auf, welche die ersten vier Kurven in verschiedenen Farben zeichnet. Beschreiben Sie, wie mit zunehmender Tiefe jedes Linienelement ersetzt wird.

c Öffnen Sie den Quelltext der Klasse KURVE und erstellen Sie für den Aufruf von *SchrittAusführen* je ein Sequenzdiagramm mit den Werten 0 und 2 für den Parameter tiefe analog zum Bild rechts. Geben Sie begründet an, wie die Methodenaufrufe zu Ihren Beobachtungen von b) passen.

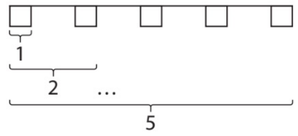


d Geben Sie an, welche Bedeutung die bedingte Anweisung in der Methode *SchrittAusführen* hat.

e Für Schnelle: Ergänzen Sie in der Methode *KurveZeichnen* Aufrufe der Methode *SchrittAusführen* so, dass eine zu den waagrechten Achsen symmetrische Figur entsteht (Tipp: benötigt werden u. a. insgesamt vier Aufrufe von *SchrittAusführen*).

Immer einfacher

Um die nebenstehend schematisch gezeichnete Perlenkette mit 5 Perlen zu zeichnen, kann man eine Kette mit 4 Perlen zeichnen und dann ein weiteres Element (Schnur und Perle) anfügen. Diese Verkürzung führt man fort, bis schließlich die Kette nur noch eine Perle (ein Rechteck) hat, die man sofort zeichnen kann.



Algorithmische Umsetzung

Die Konstruktionsbeschreibung der Perlenkette lässt sich direkt als Algorithmus formulieren. In Pseudocode lautet die Methode, welche das Erstellen steuert, wie folgt:

```
methode SchrittAusführen(anzahl: GANZZAHL)
    falls anzahl == 1                          ← Abbruchbedingung
    dann
        QuadratZeichnen(10)
    sonst
        SchrittAusführen(anzahl - 1)
        turtle.Gehen(30)                       ← Rekursionsschritt
        QuadratZeichnen(10)
    endefalls
endemethode
```

Es kann auch ein situationsabhängiger Methodenname wie KettenGliedZeichnen verwendet werden.

→recurrere: lat. zurücklaufen

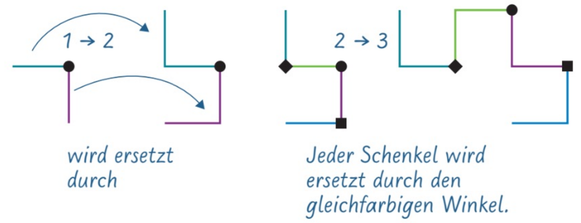
Rekursives Vorgehen

An diesem Algorithmus fällt sofort auf, dass sich die Methode *SchrittAusführen* für jeden Verfeinerungsschritt selbst aufruft. In der Informatik nennt man einen derartigen Selbstaufruf **rekursiven Aufruf**; das Verfahren mit rekursiven Aufrufen heißt →**Rekursion**.

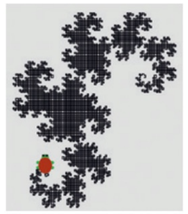
Das Prinzip jeder Rekursion besteht darin, dasselbe Verfahren auf eine gleichartige Problemsituation anzuwenden (**Rekursionsschritt**), die sich durch die Anwendung immer stärker vereinfacht (hier wird die noch zu zeichnende Restkette immer mehr verkürzt). Im Verlauf der Abarbeitung wird eine **Abbruchbedingung** erfüllt (hier: `anzahl == 1`); das **Rekursionsende** ist erreicht und die Lösung der einfachen Situation (hier: das Zeichnen einer Perle) kann direkt erfolgen. Ruft sich eine Methode wie in diesem Beispiel an nur einer Stelle auf, spricht man von einer **linearen Rekursion**.

Selbstähnliche Kurven

Eine bekannte, selbstähnliche Figur ist die sogenannte Drachenkurve. Konstruiert wird sie sehr einfach: Grundelement ist ein 90°-Winkel (links: Stufe 1). Bei jeder Rekursionsstufe (1 → 2, 2 → 3) wird jeder Schenkel durch einen 90°-Winkel ersetzt.



Dabei hat der Winkel für die Ersetzung des zweiten Schenkels eine umgekehrte Drehrichtung.



Screenshot der Drachenkurve

Algorithmische Umsetzung

Die Konstruktionsbeschreibung der Drachenkurve lässt sich direkt als Algorithmus formulieren. Der Wert des Parameters stufe wird immer kleiner, bis man auf Stufe 1 den Winkel zeichnen kann. In Pseudocode lautet die Methode, welche die Verfeinerungen steuert, wie folgt:

```
methode SchrittAusführen(drehtLinks: WAHRHEITSWERT, stufe: GANZZAHL)
    falls stufe == 1
    dann
        turtle.Gehen(50)
        Drehen(drehtLinks)                     ← Die Methode Drehen dreht die Turtle um 90° nach links, wenn der Parameter WAHR ist, sonst 90° nach rechts.
        turtle.Gehen(50)
    sonst
        SchrittAusführen(wahr, stufe - 1)
        Drehen(drehtLinks)
        SchrittAusführen(falsch, stufe - 1)
    endefalls
endemethode
```

Bei dem obigen Beispiel ruft sich die Methode *SchrittAusführen* immer an zwei Stellen auf (oder gar nicht), man spricht hier von einer **verzweigten Rekursion**.

Ein Verfahren heißt **rekursiv**, wenn es sich selbst auf immer einfachere Teile der Anfangssituation anwendet, bis die Situation am Ende so einfach ist, dass sie direkt gelöst werden kann.
Rekursive Methoden rufen sich selbst direkt oder indirekt so lange auf (**Rekursionsschritt**), bis eine **Abbruchbedingung** erreicht ist, so dass die Selbstaufrufe zu einem Ende kommen **(Rekursionsende).**
Je nach Anzahl der Selbstaufrufe spricht man von **linearer** (ein Aufruf) oder **verzweigter** (zwei oder mehr Aufrufe) **Rekursion**.

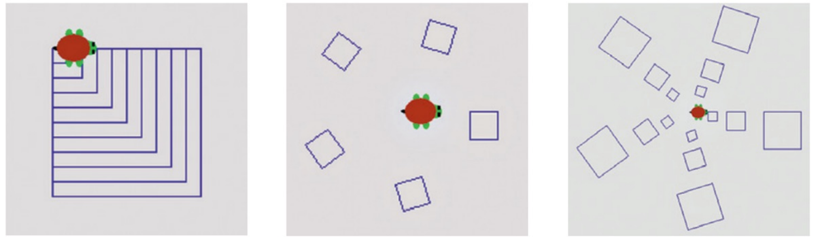
Aufgaben

1 Rekursion mit einer Turtle-Grafik

Mit Objekten der Klasse TURTLE lassen sich rekursive Muster einfach zeichnen. Das Vorlagenprojekt enthält bereits eine Klasse REKURSIVEGRAFIK, in deren Konstruktor ein Turtleobjekt erzeugt wird. Weiter enthält sie eine Methode *QuadratZeichnen(länge)*, die an der aktuellen Turtleposition ein Quadrat mit Seitenlänge länge zeichnet.

a Implementieren Sie die rekursive Methode *PerlenketteZeichnen(anzahl)* analog zum Lehrtext, welche abhängig vom Eingabewert eine Kette mit Quadraten als Perlen zeichnet, deren Anzahl durch den Parameter anzahl festgelegt wird.

b Erstellen Sie selbst eine rekursive Methode *QuadratmusterZeichnen(anzahl)*, die anzahl Quadrate mit den Seitenlängen 10, 20, 30, …, anzahl*10 zeichnet (s. Abb. unten links).



c Erstellen Sie eine rekursive Methode *MusterZeichnen(gesamtanzahlQuadrate, nochZuZeichnendeQuadrate)*, die auf einer imaginären Kreislinie Quadrate mit einer Seitenlänge von 20 zeichnet. Der erste Parameter wird zur Berechnung des Drehwinkels benötigt. Die Abbildung oben Mitte ist das Ergebnis des Methodenaufrufs *MusterZeichnen(5, 5)*. Testen Sie mit mindestens 2 unterschiedlichen Quadratanzahlen.

d Überlegen Sie sich selbst andere Muster, die sich rekursiv implementieren lassen, und setzen Sie sie um. Ein mögliches Beispiel ist in der Abbildung oben rechts dargestellt.

2 Palindromüberprüfung

Ein Palindrom ist eine Zeichenkette, die vorwärts wie rückwärts gelesen identisch ist. Zum Beispiel sind „kajak" und „hannah" Palindrome, aber „boot" und „hanna" nicht. Auch der Satz „reit nie ein tier" ist ein Palindrom.

Schreiben Sie eine rekursive Methode *IstPalindrom(text)*, die für eine eingegebene Zeichenkette überprüft, ob es sich um ein Palindrom handelt. Testen Sie geeignet.

Hinweis: Die einzelnen Programmiersprachen bieten Methoden an, um auf die einzelnen Zeichen einer Zeichenkette zuzugreifen oder deren Länge anzugeben.

3 Rekursives Problemlösen

Folgende drei Leitfragen helfen häufig beim rekursiven Problemlösen:

- Wie lautet eine einfachere Version des gegebenen Problems? (Rekursionsschritt)
- Worin besteht der direkt lösbare Fall? (Rekursionsende)
- Wodurch wird gewährleistet, dass der direkt lösbare Fall in endlich vielen Schritten erreicht wird? (Abbruchbedingung, die erreicht werden muss)

a Beschreiben Sie ein rekursives Vorgehen, um einen Papierstreifen beliebiger Länge in 8 (bzw. 32) gleich große Teile zu teilen.

b Begründen Sie, dass das rekursive Vorgehen aus a) nicht für das Teilen in 7 gleich große Teile verwendet werden kann.

4 Rekursive mathematische Funktionen: Fakultätsfunktion

Die Fakultätsfunktion n! berechnet das Produkt $1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$. Sie lässt sich auch rekursiv einfach vereinbaren.

$$n! = \begin{cases} 1 & \text{für } n == 1 \\ (n-1)! \cdot n & \text{für } n > 1 \end{cases}$$

a Erstellen Sie in einer sonst leeren Klasse eine rekursive Methode *FakultätBerechnen(n)*, welche n! berechnet. Testen Sie mit verschiedenen Werten.

b Erstellen Sie in der gleichen Klasse eine Methode *FakultätBerechnen2(n)*, welche n! iterativ (mit einer Wiederholung) berechnet.

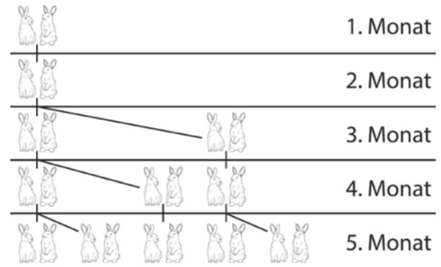
c Vergleichen Sie durch Zählung der benötigten Rechenschritte für beide Lösungen, wie die benötigte Rechenzeit von der Größe der Zahl n abhängt. Vergleichen Sie auch den Programmaufwand beider Methoden und erläutern Sie kurz, welche Implementierung Sie verwenden würden.

d Für Schnelle: Bei endlicher Stellenzahl ist die Menge der Zahlen n, für welche die Fakultät berechnet werden kann, sehr begrenzt. Bei heute üblichen 64-Bit-Systemen liegt das Maximum bei 20! = 2432902008176640000.

Erstellen Sie eine Methode *FakultätBerechnen3(n)*, welche n! durch Mehrfachauswahl über die möglichen Werte für n berechnet, bei denen der Wert von n! auf 64-Bit-Systemen darstellbar ist. Geben Sie auch hier wieder an, wie die benötigte Rechenzeit von der Größe der Zahl n abhängt, und begründen Sie daraus, dass dieser Weg den beiden in a) und b) erstellten Methoden vorzuziehen ist.

5 Rekursive mathematische Funktionen: Die Fibonacci-Funktion

Der italienische Mathematiker Leonardo Fibonacci (eigentlich Leonardo da Pisa) ist allgemein bekannt durch die nach ihm benannte Fibonacci-Folge. Diese Folge ist unter anderem die Lösung folgender Aufgabe: „Ein Kaninchenpaar zeugt, wenn es älter als zwei Monate ist, pro Monat ein weiteres Kaninchenpaar. Wie viele Kaninchenpaare gibt es nach n Monaten, wenn man mit einem neugeborenen Paar beginnt und kein Kaninchen in dieser Zeit stirbt?"



a In den ersten beiden Monaten ist nur das Ursprungspaar vorhanden. Im dritten Monat kommt ein neugeborenes Paar hinzu. Im vierten Monat ein weiteres, im fünften Monat kommen zwei neue Paare hinzu. Die Anzahlen für die ersten Monate sind also 1, 1, 2, 3, 5. Setzen Sie die Folge der Anzahlen um drei weitere Werte fort.

b Mathematisch lässt sich die Fibonacci-Funktion f(n) wie im Kasten rechts beschreiben.

$$f(n) = \begin{cases} 1 & \text{falls } n == 1 \\ 1 & \text{falls } n == 2 \\ f(n-1) + f(n-2) & \text{falls } n > 2 \end{cases}$$

Erstellen Sie in einer sonst leeren Klasse eine rekursive Methode *FibonacciBerechnen(n)*, welche die n-te Fibonaccizahl nach den Vorgaben des Kastens oben berechnet. Lassen Sie zu Beginn der Methode den Parameter auf die Konsole ausgeben, um die rekursiven Aufrufe zu dokumentieren. Überprüfen Sie damit Ihre Fortsetzungen aus a).

c Geben Sie an, um welche Art von Rekursion es sich bei der Implementierung von *FibonacciBerechnen* handelt. Geben Sie weiter an, welche Besonderheit die Terminierung des rekursiven Aufrufs von *FibonacciBerechnen* hat und warum diese Sondersituation nötig ist.

6 Rekursive mathematische Funktionen: Größter gemeinsamer Teiler

Der größte gemeinsame Teiler zweier Zahlen zahl1 und zahl2 lässt sich rekursiv sehr einfach berechnen:

```
methode GGTBerechnen(zahl1: GANZZAHL, zahl2: GANZZAHL) -> GANZZAHL
    falls zahl2 == 0
    dann
        return zahl1
    sonst
        return GGTBerechnen(zahl2, zahl1 % zahl2)
    endefalls
endemethode
```

a Setzen Sie fort: *GGTBerechnen(24, 15) = GGTBerechnen(15, 9) = …*

b Implementieren Sie die Methode *GGTBerechnen*. Testen Sie mit mehreren geeigneten Beispielen (insbesondere auch, dass der GGT von zahl1 und zahl2 1 ist oder dass zahl1 ein Teiler von zahl2 ist).

c Geben Sie an, warum die Methode *GGTBerechnen* linear rekusiv ist.

d Im Fall der Methode *GGTBerechnen* spricht man auch von endrekursiv. Erläutern Sie diese Bezeichnung.

7 Rekursive mathematische Funktionen: Die Ackermann-Funktion

Um die Vermutung zu widerlegen, dass jede berechenbare Funktion primitiv-rekursiv ist (sich also immer auch mit einer Iteration berechnen lässt), veröffentlichte Wilhelm Ackermann 1928 die nach ihm benannte Ackermann-Funktion. 1935 veröffentlichte Rósza Péter eine vereinfachte Version, die Ackermann-Péter-Funktion:

$$\text{Acker}(n, m) = \begin{cases} m & \text{falls } n == 0 \\ \text{Acker}(n-1,1) & \text{falls } m == 0 \\ \text{Acker}(n-1, \text{Acker}(n, m-1)) & \text{sonst} \end{cases}$$

a Erstellen Sie in einer sonst leeren Klasse eine rekursive Methode *Ackermann-Berechnen*, die die Ackermann-Péter-Funktion berechnet. Testen Sie zunächst mit sehr kleinen, natürlichen Zahlen.

b Ergänzen Sie Ihre Lösung aus a) um Ausgaben der Parameterwerte zu Beginn der Methode, um die Art der Aufrufe zu dokumentieren.
Notieren Sie auch eine Tabelle, die den ursprünglichen Parameterwerten die Gesamtzahl der Methodenaufrufe zuordnet. Erzeugen Sie 10 Einträge mit kleinen Werten für m und n. Begründen Sie aus Ihrer Tabelle, warum die Ackermann-Péter-Funktion auch gern zum Testen der Rechengeschwindigkeit von Computern verwendet wurde.

c Für Schnelle: Ergänzen Sie Ihre Lösung aus b) so, dass mit zunehmender Aufruftiefe die Ausgabe der Parameterwerte immer weiter eingerückt wird. Lassen Sie mit dieser Lösung zwei sogenannte Aufrufbäume für kleine Werte von n und m zeichnen.

8 Mathematische Spielerei: Gerade oder ungerade

Die Funktion *IstGerade(n)* soll genau dann WAHR zurückgeben, wenn die Zahl n gerade ist. Eine interessante rekursive Definition für diese Berechnung ist:

a Erstellen Sie in einer sonst leeren Klasse beide Methoden und testen Sie mit mindestens vier geeigneten Werten.

$$\text{Gerade}(n) = \begin{cases} \text{wahr} & \text{falls } n == 0 \\ \text{Ungerade}(n-1) & \text{falls } n > 0 \end{cases}$$

$$\text{Ungerade}(n) = \begin{cases} \text{falsch} & \text{falls } n == 0 \\ \text{Gerade}(n-1) & \text{falls } n > 0 \end{cases}$$

b Erstellen Sie das Sequenzdiagramm für den Aufruf *IstGerade(3)*.

c Eine übliche Art zu testen, ob eine Zahl n gerade ist, ist zu prüfen, ob die Division n/2 den Rest 0 oder 1 hat. Erstellen Sie in Ihrer Klasse eine weitere Methode *IstGerade2(n)*, welche den Rest der Division durch 2 nutzt. Geben Sie für beide Ansätze die Anzahl der Methodenaufrufe für den Eingabewert 5 an.

9 Türme von Hanoi

Auf der Weltausstellung 1889 in Paris wurde das Spiel „Die Türme von Hanoi“ vorgestellt. Wenn der Scheibenturm auf eine der anderen Säulen umgeschichtet werden soll, gelten die folgenden Regeln:

- Die Scheiben dürfen nur einzeln von einer Säule zu einer anderen transportiert werden.
- Es darf niemals eine Scheibe mit einem größeren Durchmesser auf eine Scheibe mit einem kleineren Durchmesser gelegt werden.

a Skizzieren Sie auf Papier eine Lösung für einen Anfangsturm mit zwei Scheiben und für einen mit drei Scheiben. Geben Sie an, wie sich die Lösung für drei Scheiben auf die Lösung für zwei Scheiben zurückführen lässt.

b Geben Sie an, wie sich allgemein die Lösung für n Scheiben auf die Lösung für n–1 Scheiben zurückführen lässt.

c Der folgende rekursive Algorithmus löst das Problem für beliebige Turmhöhen. Die Methode *TurmVerschieben* bewegt die oberste Platte des Turms mit der Nummer „von“ auf den Turm mit der Nummer „nach“, die Türme haben die Nummern 0, 1 und 2.

```
methode TurmVerschieben(anzahlScheiben, von, nach)
    falls anzahlScheiben > 0
    dann
        frei = 3 - von - nach
        TurmVerschieben(anzahlScheiben - 1, von, frei)
        Bewegen(von, nach)
        TurmVerschieben(anzahlScheiben  - 1, frei, nach)
    endefalls
endemethode
```

Erklären Sie die Funktionsweise unter Zuhilfenahme von b).
Erläutern Sie, was mit einem Stapel der Höhe 0 oder 1 passiert.

d Im Vorlagenprojekt sind bereits Methoden zur graphischen Darstellung des Verschiebevorgangs gegeben. Implementieren Sie die Methode *TurmVerschieben* und testen Sie den Algorithmus mit verschiedenen Höhen, aber nicht höher als 10 Scheiben.

e Für Schnelle: Bei Höhen über 10 dauert das Verschieben schon bald sehr lange. Zählen Sie für Türme der Höhen 1, 2 und 3, wie viele Verschiebungen insgesamt nötig sind. Erstellen Sie daraus eine Vermutung, wie viele Verschiebungen für einen Turm der Höhe n nötig sind, und begründen Sie damit die lange Laufzeit für hohe Türme.

*Eine linear rekursive Funktion heißt **endrekursiv**, wenn nach dem rekursiven Aufruf keine Anweisungen mehr folgen.*

10 Rekursion bei Listen und Bäumen

Bei der Bearbeitung von Listen und Bäumen werden sehr oft rekursive Methoden verwendet, da diese Datenstrukturen selbst rekursiv definiert sind.

a Geben Sie an, um welche Art von Rekursion es sich bei den Methoden der Listen handelt und welche Art bei den Bäumen vorliegt. Begründen Sie den Unterschied aus der Definition der zugehörigen Datenstrukturen.

b Die Methode *LängeGeben()* einer Liste ist endrekursiv, daher kann die Länge der Liste sicher auch iterativ (mit einer Wiederholung) berechnet werden. Implementieren Sie in dem Vorlagenprojekt eine Methode *LängeGeben2()*, welche iterativ arbeitet. Vergleichen Sie beide Implementierungen und geben Sie mit kurzer Erläuterung an, welche Sie bevorzugen würden.

c Geben Sie begründet an, warum Sie mit einer Laufzeitmessung (Messung der Zeit vom Programmstart bis zum Beenden des Programms) nicht feststellen könnten, welche der beiden Implementierungsarten vorliegt.

d Für Schnelle: Zeigen Sie, z. B. durch Angeben eines Umbauschemas, dass sich jede endrekursive Funktion auch iterativ berechnen lässt.

11 Selbstähnliche Figuren: Sierpinski-Dreieck

Ausgangssituation: weißes Dreieck als Hintergrund

eine Unterteilung

zwei Unterteilungen

drei Unterteilungen

Das gegebene Dreieck wird mit jedem Rekursionsschritt in drei Dreiecke mit halber Seitenlänge in den Ecken unterteilt. Ist das Rekursionsende erreicht, wird das Dreieck schwarz gezeichnet. Die auf der Spitze stehenden weißen Mitteldreiecke sind der noch sichtbare Teil des Hintergrunds.

a Ergänzen Sie in der Klasse SIERPINSKIDREIECK den Rumpf der Methode *SchrittAusführen*.

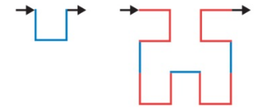
b Geben Sie an, um welche Art der Rekursion es sich hier handelt.

c Zeichnen Sie die Figuren für verschiedene Rekursionsstufen.

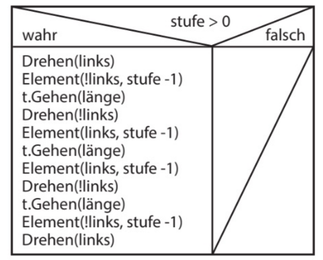
12 Selbstähnliche Figuren: Die Hilbert-Kurve

Die nach dem deutschen Mathematiker David Hilbert benannte Kurve ist die erste bekannte Kurve (ein stetiger, d. h. nicht unterbrochener, eindimensionaler Linienzug), die nach und nach die zweidimensionale Grundfläche völlig ausfüllt.

Bei jedem Rekursionsschritt werden vor, zwischen und nach den drei Linien eines Elements (blau) neue Elemente eingefügt (rot), allerdings mit unterschiedlicher Drehrichtung. Auch muss sich die Turtle zu Beginn und am Ende eines Elements drehen, um die jeweils angegebene Blickrichtung zu haben.



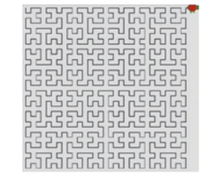
Für die rekursive Methode *Element(links,stufe)* ergibt sich daraus das nebenstehende Struktogramm.



a Verifizieren Sie mit Bleistift und kariertem Papier für die Stufen 1 und 2, dass das Struktogramm die gezeigten Grundelemente liefert. Die Turtle schaut zu Beginn nach rechts, die Länge entspricht zwei Kästchen.

b Geben Sie an, von welchem Typ die Rekursion ist.

c Ergänzen Sie im Vorlagenprojekt den Rumpf der Methode *Element* und lassen Sie die Kurve für verschiedene Tiefen ausgeben. Geben Sie auch an, ab welcher Rekursionsstufe sich erahnen lässt, dass die Kurve für unendliche Rekursionstiefe die Fläche füllen könnte.



Screenshot der Hilbert-Kurve

13 Selbstähnliche Figuren: Pythagoras-Baum

a Erzeugen Sie für verschiedene Rekursionstiefen je ein Objekt der Klasse PYTHAGORAS-BAUM und erläutern Sie daraus, wie die Figur zu ihrem Namen kommt.

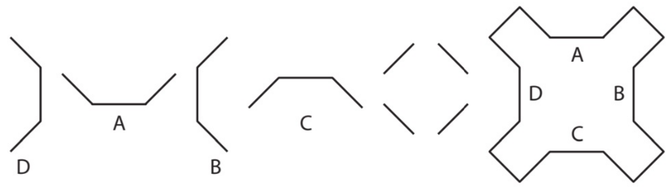
b Beschreiben Sie in knappen Worten, aus welchen Grundelementen der Baum besteht und wie diese mit jeder Rekursionsstufe ergänzt werden.

c Die Attribute a und b bestimmen das Seitenverhältnis der Katheten. Ändern Sie die vorgegebenen Werte und erkunden Sie, wie sich der Baum dadurch ändert.

14 Selbstähnliche Figuren: Sierpinski-Kurve

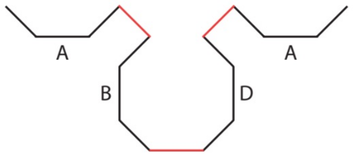
Die Sierpinski-Kurve ist eine raumfüllende Kurve, d. h. sie füllt das umgebende Quadrat mit zunehmender Rekursionstiefe immer mehr aus.

Die Grundfigur besteht aus vier Grundelementen und vier Verbindungsstrecken, die zur rechts stehenden Grundfigur verbunden werden.



Mit jedem Rekursionsschritt werden die Grundelemente A, B, C und D durch mehrere Ausgaben ihrer selbst ersetzt; unten ist das exemplarisch für A dargestellt.

wird ersetzt durch:



Die vier Ersetzungen für das Grundelement B sind entsprechend B-C-A-B.

a Geben Sie die Ersetzungen für die Grundelemente C und D analog an.

b Ergänzen Sie das Projekt zum Zeichnen dieser Kurve und prüfen Sie die Aussage, dass die Kurve allmählich das ganze umgebende Quadrat ausfüllt, auf Plausibilität.

3.2 Anwendung der Rekursion: Backtracking

In einer Quizshow haben die Kandidaten u. a. die Aufgabe, Paletten unterschiedlicher Höhe und mit unterschiedlichem Gewicht so aufeinander zu stapeln, dass bei einer Maximalhöhe von 150 cm der Palettenstapel ein möglichst hohes Gesamtgewicht bekommt.

a Erzeugen Sie ein Objekt der Klasse RAHMEN, drücken Sie die Starttaste, beobachten Sie, in welcher Reihenfolge die verschiedenen Möglichkeiten durchprobiert werden, und formulieren Sie die Strategie des Programms. (Tipp: Beobachten Sie die Pfeile am Vorratsstapel.)

b Lesen Sie den Quelltext der Methode *Schritt* der Klasse STAPELAUFGABE. Beschreiben Sie knapp, welchen Teil der Lösung diese Methode leistet und wie die Rekursion die Lösung vervollständigt.

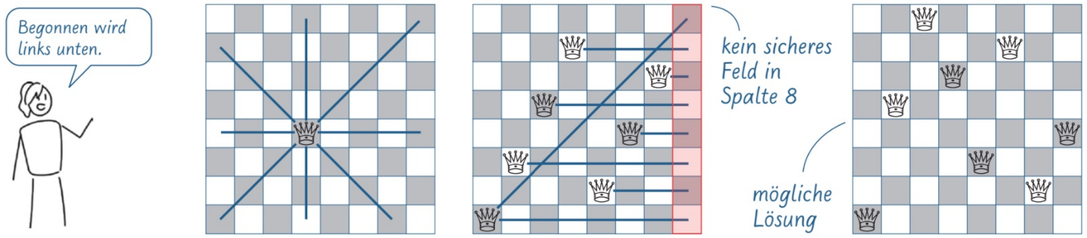
c Mit den gegebenen Paletten ist noch eine zweite, gleichwertige Lösung möglich. Geben Sie diese Lösung an (genaues Beobachten der Programmausführung oder Ausprobieren).

d Verändern Sie die Reihenfolge der Paletten im Palettenvorrat so, dass diese andere Lösung zuerst gefunden wird.

e Für Schnelle: Legen Sie nun eigene Palettenvorräte an. Vermuten Sie zunächst die dafür optimale Lösung und vergleichen Sie Ihre Lösung mit dem Ergebnis des Programms.

Eine klassische Frage

Der bayerische Schachmeister Max Bezzel stellte 1848 in der Berliner Schachzeitung die Frage, auf wie viele Arten man acht Damen so auf einem Schachbrett unterbringen kann, dass sie sich nicht gegenseitig schlagen können. Da eine Dame als mächtigste Figur senkrecht, waagrecht und diagonal schlagen kann (siehe Bild unten links), kann sicher in jeder Spalte des Schachbretts nur eine Dame stehen.



Teile und herrsche (und finde so die Lösungen)

Zum Finden aller möglichen Lösungen teilt man das Feld in seine Spalten und geht dann mit einer rekursiven Methode *SchrittAusführen* Spalte für Spalte weiter, bis man eine Lösung findet oder keine Möglichkeit mehr existiert, eine Dame in der aktuellen Spalte zu positionieren. Um keine Möglichkeit auszulassen, probiert man in jeder Spalte z. B. Zeile für Zeile, bis ein sicheres Feld (d. h. ein Feld, das von keiner der schon gesetzten Damen geschlagen werden kann) gefunden ist. So kann man ohne Problem die ersten sieben Damen positionieren (Bild oben Mitte). Die Dame in der 8. Spalte kann aber in dieser

SchrittAusführen(aktuelle Spalte)

```
solange es noch sichere Felder in der aktuellen Spalte gibt
    Dame im nächsten sicheren Feld positionieren
    wenn aktuelle Spalte ist 8
        dann Lösung ausgeben
        sonst SchrittAusführen(aktuelle Spalte + 1)
    Dame vom Feld entfernen
```


Konstellation nicht mehr platziert werden; alle acht Felder können von einer der bisherigen Damen geschlagen werden. Es muss also in den vorigen Spalten nach weiteren Konstellationen gesucht werden, was durch den rekursiven Ansatz kein Problem ist. Erfolgt kein rekursiver Aufruf von *Schritt-Ausführen* mehr (entweder, weil eine Lösung gefunden wurde oder weil es in der aktuellen Spalte keine sicheren Felder mehr gibt), wird die zuletzt gesetzte Dame wieder vom Feld genommen und ein anderes Feld für sie gesucht. Führt dies nicht zum Erfolg, wird nach ihr auch die vorletzte gesetzte Dame entfernt, neu positioniert usw. Dieses mögliche „Zurückgehen auf der eigenen Spur" gibt dem Vorgehen seinen (englischen) Namen: **Backtracking**. Das Bild S. 96 rechts zeigt die erste Lösung.

Möglichst viel reinpacken

Oft benötigt man aber nicht alle Lösungen eines Problems, sondern nur die beste: Für eine Wanderung packt man einen Rucksack mit möglichst vielen Essensvorräten. Aber: Man darf dabei ein Maximalgewicht (z. B. 30 kg) nicht überschreiten, um den Rucksack nicht zu schwer zu machen. Dabei muss man beachten, dass die verschiedenen Vorräte einen unterschiedlichen Nährwert pro kg Gewicht haben.

Lösung: Systematisch den „vollsten" Rucksack finden

Um die **optimalen** Vorräte auszuwählen, muss man systematisch alle Kombinationsmöglichkeiten durchprobieren. Zu Beginn hat man noch keine Lösung, der beste Nährwert ist 0. Bei jeder Kombination (Lösung), die man mitnehmen kann, prüft man, ob sie einen höheren Nährwert hat als die Beste, die man bisher ermittelt hat. Wenn ja, wird sie als neues Optimum gespeichert.

Um aus der nebenstehenden Menge an Vorräten die optimale Beladung des Rucksacks zu finden, ordnet man sie in einem Feld an. Im ersten Rekursionsschritt geht man die Vorräte ab Nummer 0 durch. Jeder Rekursionsschritt testet dann die Vorräte ab dem nächsten noch nicht verwendeten Vorratsstück.

Nummer	0	1	2	3	4	5	6	7
Gewicht [kg]	5	7	4	3	10	12	13	8
Nährwert [MJ]	40	35	12	30	25	84	78	80

NächsterSchritt(startIndex: GANZZAHL, gesamtGewicht: GANZZAHL, gesamtNährwert: GANZZAHL)

```
zähle index von startIndex bis vorräte.längeGeben() - 1
    gewicht = gesamtGewicht + vorräte.ElementGeben(index).GewichtGeben()
    wenn gewicht <= 30
        dann wahr
        sonst falsch
    nährwert = gesamtNährwert + vorräte.ElementGeben(index).NährwertGeben()
    wenn nährwert > besterNährwert
        dann wahr
        sonst falsch
    neue optimale Lösung speichern
    NächsterSchritt(index + 1, gewicht, nährwert)
```

Das Rekursionsende ist erreicht, wenn entweder das Gesamtgewicht zu groß wird oder der Maximalindex des Vorratsfeldes erreicht ist.

Mit Hilfe des **Backtracking**-Verfahrens können systematisch **alle Lösungen** einer gegebenen Situation rekursiv gesucht werden.
Dabei versucht man, in geeigneten Schritten auf die Lösung zuzugehen. Sobald man erkennt, dass ein Weg nicht mehr weitergeführt werden kann, geht man unter Verwendung der Rekursion auf der eigenen Spur zurück, bis ein neuer Weg in Richtung Lösung möglich ist.
Mit dem Backtracking-Verfahren kann auch die **optimale Lösung** gesucht werden.

Aufgaben

1 Sechs Damen

a Versuchen Sie von Hand (z. B. mit Bleistift und Kästchenpapier), auf einem Schachbrett der Größe 6 × 6 sechs Damen so aufzustellen, dass keine die andere schlagen kann.
Tipp: Beginnen Sie für die 1. Dame in Zeile 2.

b Für Schnelle: Suchen Sie alle 4 möglichen Lösungen.

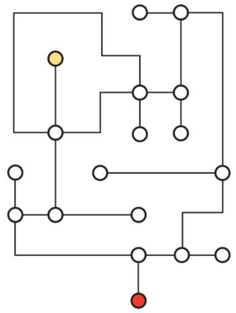
2 Rucksackproblem manuell

Lösen Sie das Rucksackproblem mit Bleistift und Papier für die nebenstehende Vorratsmenge und die Vorgabe, dass der Rucksack maximal 10 kg wiegen darf.

Nummer	0	1	2	3
Gewicht [kg]	5	7	4	3
Nährwert [MJ]	40	35	12	30

3 Höhlenlabyrinth

Höhlenforscher wissen, dass in dem rechts schematisch dargestellten Höhlensystem eine Kammer mit wunderbaren Tropfsteinen existiert (gelb markiert). Die weißen Kreise (Knoten) markieren Kammern, die Verbindungslinien (Kanten) die existierenden Gänge. Der rote Kreis markiert den Eingang.



a Finden Sie per Backtracking von Hand einen Weg vom Eingang zur Tropfsteinkammer für die Forscher.
Tipp: Geben Sie den Knoten Nummern und legen Sie zu Beginn fest, in welcher Reihenfolge Sie in jeder Kammer die Gänge besuchen wollen.

b Geben Sie eine Möglichkeit an, wie ein Forscherteam das Backtracking in der Höhle durchführen könnte.

4 Acht Damen

Ergänzen Sie wie in den Teilaufgaben beschrieben schrittweise das gegebene Projekt zur vollständigen Lösung.

a Öffnen Sie den Quellcode der Klasse DAMEN. Beschreiben Sie in kurzen Worten die Arbeitsweise der Methoden *LösungAusgeben* und *IstFeldBedroht*.

b Ergänzen Sie den Rumpf der Methode *SchrittAusführen* gemäß des im Lehrtext angegebenen Struktogramms unter Verwendung der vorhandenen Methoden.

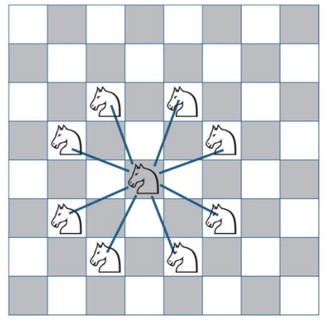
c Für Schnelle: Ändern Sie den Konstruktor der Klasse DAMEN so ab, dass Schachbretter beliebiger Größe untersucht werden können. Bestimmen Sie damit, ab welcher Mindestgröße überhaupt Lösungen möglich sind.

d Für ganz Schnelle: Die Klasse DAME erlaubt, eine Dame auf der Zeichenfläche darzustellen. Nutzen Sie diese Klasse zusammen mit einem Objekt der Klasse SCHACHBRETT, um die Lösungen auf der Zeichenfläche optisch ansprechend darzustellen.
Tipp: Sie müssen nach dem Zeichnen einer Lösung auf einen Tastendruck o. Ä. warten, um die Lösung auch betrachten zu können.

5 Für Schnelle: Rösselsprung

Die folgende Aufgabe kommt auch aus dem Schachspiel. Ein Springer soll, von einem beliebigen Feld startend, so über das Brett geführt werden, dass er alle Felder genau einmal betritt. Der Springer hat die bei Teilaufgabe a) gezeigten acht Zugmöglichkeiten.

a Erstellen Sie ein Programm, welches die Startposition vom Benutzer erfragt und dann einen Weg über das Brett sucht. Gehen Sie dabei wie folgt vor:



- Das Schachbrett speichert die Nummer der Springerpositionen. Die Startposition hat die Nummer 1; die Nummer 0 bedeutet „freies Feld".
- Die Methode *SchrittAusführen* beschreibt einen Zug des Springers. Die Wiederholung geht über alle 8 (prinzipiellen) Zugmöglichkeiten.
- Im Konstruktor wird die Startposition gesetzt.

Hinweis: Da die Laufzeit für ein reguläres Schachbrett sehr hoch ist, beschränken Sie die Aufgabe auf ein Brett der Größe 5 × 5 oder 6 × 6.

b Eine härtere Bedingung für die Lösung ist ein sogenannter „geschlossener Weg". Vom letzten betretenen Feld aus muss der Springer das Anfangsfeld in einem Zug erreichen können. Erweitern Sie Ihr Programm so, dass es nur „geschlossene Wege" als Lösung der Aufgaben anzeigt.

6 Das Rucksackproblem

Im Vorlagenprojekt sind die Vorratsdaten aus dem Lehrtext bereits als mögliches Beispiel vordefiniert.

a Erkunden Sie im Quelltext der Klasse RUCKSACK, mit welchen Methoden die Speicherung und Ausgabe der (bisherigen) optimalen Lösung erfolgt.

b Ergänzen Sie in dem gegebenen Projekt den Rumpf der Methode *SchrittAusführen* gemäß des Lehrtextes.

c Überlegen Sie sich mindestens zwei weitere Anwendungssituationen, deren Lösung dem Rucksackproblem entspricht.

d Für Schnelle: Implementieren Sie eines der in c) gefundenen Beispiele mit geeigneten Daten.

7 Sudoku

Beim Sudoku-Rätsel müssen in ein Feld der Größe 9 × 9 die Zahlen von 1 bis 9 nach festen Regeln eingetragen werden: pro Zeile, pro Spalte und pro Teilquadrat müssen alle Zahlen genau einmal stehen.
Dafür gibt es sehr viele Lösungen. In den gestellten Rätseln sind daher bereits so viele Werte vorgegeben, dass genau eine Lösung möglich ist.

		2						
6				3				
4		9	1	6	7			3
	1							
		8			4	3		2
						4	7	
	8			9				
				4	3			8
			5	7		9		

a Legen Sie das Sudokufeld als 2-dimensionales Feld an; die Elemente sollen wieder Felder sein mit den Werten, die auf dieser Position noch möglich sind. Zu Beginn sind in allen Elementen alle Zahlen möglich.

b Ergänzen Sie den Rumpf der Methode *FeldelementFixieren(zeile, spalte, wert)*, welche auf dem Element in der angegebenen Zeile und Spalte den gegebenen Wert als einzigen Wert einträgt und aus allen betroffenen Elementen (Zeile, Spalte, kleines Quadrat) diesen Wert aus dem Feld der noch möglichen Werte löscht.

c Ergänzen Sie den Rumpf der Methode *FeldAusgeben*, welche das Feld auf die Konsole ausgibt. Ist für ein Feld nur noch ein Wert möglich, so wird dieser angezeigt, andernfalls wird ein Strich „-" ausgegeben.

d Erstellen Sie nun die Methode *SchrittAusführen(zeile, spalte)*, die für jedes Feldelement alle Werte durchprobiert.
Achtung: Ehe Sie mit *FeldelementFixieren* einen der möglichen Werte fixieren, müssen Sie eine Kopie des Feldes erstellen, um für den nächsten Versuch wieder die Ausgangssituation herstellen zu können. Verwenden Sie zum Erstellen der Kopie und zum Wiederherstellen des vorigen Zustands die Methode *Kopieren*.

e Die Methode *Vorbesetzen* richtet die im Bild oben gegebene Anfangsstellung ein. Für diese Anfangsstellung existiert genau eine Lösung. Testen Sie damit Ihre Implementierung. Entfernen Sie einzelne Vorbesetzungen aus der Anfangsstellung und führen Sie Ihren Algorithmus damit aus. Verifizieren Sie, dass alle angezeigten Lösungen den Sudokuregeln entsprechen.

8 Backtracking erklärt
Erklären Sie anhand eines eigenen Beispiels Vorgehensweise und Vorteile des Backtracking-Verfahrens.

***9 Acht-Damen optimiert**
In der im Lehrtext und in Aufgabe 4 angesprochenen Lösung des 8-Damen-Problems wird die Information über die bereits positionierten Damen sehr verschwenderisch gespeichert. Eine optimierte Speicherung ist, pro Spalte nur die Zeilennummer zu speichern, in der die Dame positioniert ist.

a Ersetzen Sie das zweidimensionale Feld mit den logischen Werten durch ein eindimensionales Feld mit ganzen Zahlen und ändern Sie den Konstruktor entsprechend.

b Passen Sie die Methode *IstFeldBedroht* an die neue Speicherung der Damen an. Sie können dabei ausnutzen, dass für die Diagonalen die Summe bzw. Differenz von Spalten- und Zeilennummern konstant ist.

c Passen Sie die Methoden *SchrittAusführen* und *LösungAusgeben* an. Testen Sie die neue Implementierung.

d Begründen Sie insbesondere aus der Implementierung der Methode, dass die neue Lösung nicht nur weniger Speicherplatz benötigt, sondern auch noch schneller ist.

***10 Sudoku optimiert**
Die Elemente des Sudoku-Feldes stur ihrer Reihenfolge nach abzuarbeiten, kann, insbesondere wenn viele Lösungen möglich sind, zu langen Rechenzeiten führen. Eine gute Optimierung ist, immer mit dem Element mit den wenigsten Möglichkeiten fortzusetzen. Dabei muss man sich aber merken, welche Elemente schon bearbeitet sind, damit kein Element mehrfach ausgewählt wird. Das Rekursionsende ist dann erreicht, wenn alle 81 Elemente bearbeitet sind. Eine Sackgasse erkennt man sofort daran, dass das Element mit der geringsten Elementanzahl keine Möglichkeiten mehr enthält.
Implementieren Sie diese Optimierung.

***11 Voice Of Infohausen: Möglichst gute Lösung**
In einem Stimmenwettbewerb im Fernsehen soll einer Anzahl n von Kandidaten jeweils ein Gesangscoach zugeordnet werden. Dabei sollen die Paare Kandidat - Coach möglichst optimal zusammenpassen. Dazu erstellt jeder Kandidat eine Wunschliste mit den Werten von 1 (mag ich am wenigsten) bis n (hätte ich am liebsten) für seinen Coach; umgekehrt erstellt jeder Coach eine entsprechende Wunschliste, welchen Kandidaten er am liebsten coachen würde.
Dass dabei alle Coaches und alle Kandidaten ihren jeweils am höchsten bewerteten Wunschpartner bekommen, wäre äußerst unwahrscheinlich. Es muss eine Aufstellung (= Paarung aller Kandidaten und Coaches) gefunden werden, mit der keiner unzufrieden ist. Deshalb soll es keine Aufstellung geben, in der es einen Coach und einen Kandidaten gibt, die zusammen kein Paar bilden, obwohl sie sich gegenseitig ihrem jeweiligen Partner vorziehen würden. Solche Aufstellungen werden stabil genannt.

a Erkunden Sie die Klasse SONGWETTBEWERB und die Bedeutung der vorhandenen Attribute sowie der Methoden *Lesen*, *LösungAusgeben*, *CoachFürWunschwertGeben* und *KandidatFürCoachGeben*.

b Eine mögliche Strategie zum Finden stabiler Aufstellungen versucht, jedem Kandidaten einen geeigneten Coach zuzuordnen. Dazu wird von dem am meisten gewünschten Coach bis zum am wenigsten gewünschten Coach getestet:
- Ist der Coach noch frei?
- Bleibt nach Hinzufügen des neuen Paares die bisher gebildete Aufstellung stabil (Test mit Methode *BleibtStabil*)?

Ist dieser Kandidat der letzte mit einem Coach zu verbindende Kandidat, so ist eine Lösung gefunden, ansonsten wird rekursiv mit dem nächsten Kandidaten weiter gemacht.
Ergänzen Sie den Rumpf der Methode *SchrittAusführen* entsprechend dieser Strategie.

c Die Methode *BleibtStabil* setzt voraus, dass sie bei jedem neu zur Aufstellung kommenden Paar aufgerufen wird. Daher ist sicher, dass die bisherige Aufstellung stabil ist und nur getestet werden muss, ob das neu geplante Paar Instabilitäten verursachen würde.
In der Methode wird bereits überprüft, ob es unter den bisherigen Paaren einen Kandidaten gibt, der den aktuellen Coach gegenüber seinem zurzeit zugeordneten bevorzugen würde, und gleichzeitig der aktuelle Coach diesen Kandidaten lieber hätte als den Kandidaten, mit dem er verbunden werden soll.
Ergänzen Sie in der Methode *BleibtStabil* die Überprüfung, ob es einen Coach gibt, der den aktuellen Kandidaten lieber haben würde als den ihm zurzeit zugeordneten und den umgekehrt der aktuelle Kandidat lieber hätte als den Coach, der ihm gerade zugeordnet werden soll.
Hinweis: Sie müssen alle Coaches durchgehen und darunter nur die betrachten, die bereits einem Kandidaten zugeordnet sind.

d Wie gut eine gefundene Lösung ist, hängt davon ab, wie zufrieden die einzelnen Personen mit ihren jeweiligen Partnern sind. Ein Maß für diese Zufriedenheit ist die Rangnummer des jeweiligen Partners. Erweitern Sie die Methode *LösungAusgeben* so, dass sowohl für die Kandidaten als auch für die Coaches die Summe über die Ranglistennummer der jeweiligen Partner gebildet und ausgegeben wird („ZufriedenheitKandidaten“ bzw. „ZufriedenheitCoaches“). Zusätzlich soll die „Gesamtzufriedenheit“ als Summe dieser beiden Einzelwerte ausgegeben werden.

e Recherchieren Sie im Internet zu „Stabile Heirat“ (Achtung, nicht „Heiratsproblem“, das führt zu anderen Fragestellungen.). Beschreiben Sie die allgemeine Fragestellung hinter dieser Aufgabe und nennen Sie zwei Beispiele für die praktische Anwendung.

3.3 Alle Knoten eines Graphen besuchen: Tiefensuche

Ein Vertreter für Gastronomiebedarf möchte bei seinem Tätigkeitsbeginn in einem neuen Bezirk alle seine Kunden besuchen. Dabei kommt es ihm nicht auf einen optimalen Fahrtweg an. Im Projekt Vertreter ist ein geeignetes Vorgehen implementiert.

a Öffnen Sie das Projekt und erzeugen Sie ein Objekt der Klasse VERTRETER. Rufen Sie die Methode *ReiseStarten* zunächst mit dem Parameter „M“ für München auf. Auf Tastendruck werden weitere Knoten besucht. Nennen Sie eine Vermutung, in welcher Reihenfolge die Knoten besucht werden. Erläutern Sie dabei auch die Bedeutung der Farben „blau“ und „gelb“ in der Visualisierung.

b Verwenden Sie zwei alternative Startknoten und überprüfen Sie Ihre Kriterien aus a).

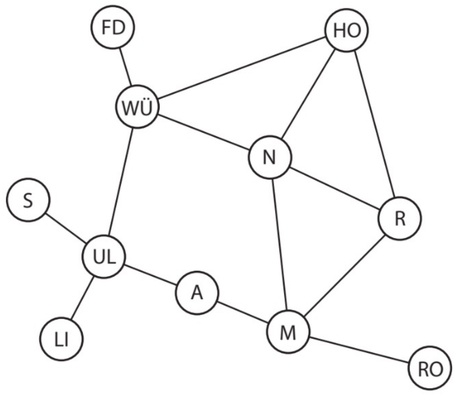
c Das Verfahren arbeitet rekursiv. Formulieren Sie aus den Beobachtungen von a) und b) einen Ansatz für eine rekursive Methode *KnotenBesuchen(aktuellerKnoten)*.
Tipp: Wenn Sie nicht weiterkommen, analysieren Sie den Quellcode der Methode *KnotenBesuchen*.

d Nennen Sie die Abbruchbedingungen der Rekursion.

e Für Schnelle: Ändern Sie in der Methode *KnotenBesuchen* die Reihenfolge der Knotenbesuche (z. B. von hinten her). Notieren Sie vor dem Ausführen der Methode die erwartete Besuchsreihenfolge und testen Sie dann.

Vorgehensweise zum Besuch aller Knoten

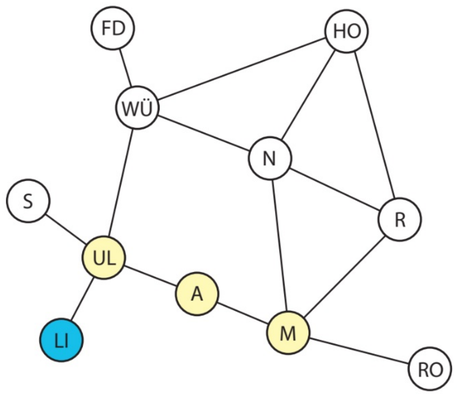
Um den aktuellen Stand der im Reiseführer empfohlenen Ziele zu überprüfen, lässt ein Verlag den zuständigen Redakteur alle beschriebenen Orte besuchen. Dabei kommt es weder auf die Länge der Route an noch auf die Frage, ob ein Ort mehrmals bereist wird; es müssen nur alle Orte besucht werden. Die Grundlage für die Planung ist das Streckennetz des abgebildeten Graphen.



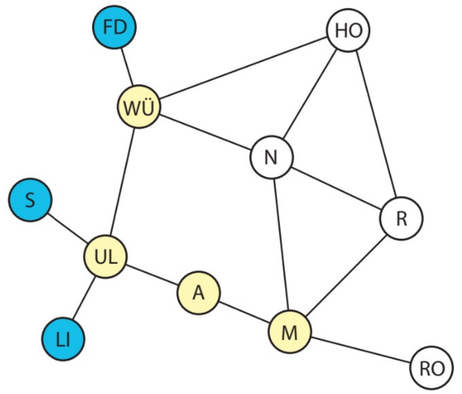
Der Redakteur beginnt seine Tour in München (M). Von München gehen vier Verbindungen ab. Für die Situation, dass an einem Ort mehrere Verbindungen abgehen, hält der Tester an jedem dieser Orte folgende Strategie ein: Wenn mehrere Wege möglich sind, dann werden die Orte immer dem Alphabet nach besucht.

So fährt er als erstes nach Augsburg. Dort stellt er fest, dass er seine Strategie ergänzen muss, weil er sonst sofort zurück nach München fahren würde: Jeder besuchte Ort wird markiert und Wege zu bereits markierten Orten werden bei der Auswahl der Verbindungen nicht berücksichtigt.

Aufgrund dieser Ergänzung fährt er nunmehr nach Ulm und dann nach Lindau. Hier geht es nicht weiter, also muss er seine Strategie



nochmals ergänzen: Wenn es an einem Ort nicht weiter geht, dann kehre zu dem Nachbarort zurück, von dem aus dieser Ort besucht wurde. Im Bild rechts sind die Orte gelb markiert, zu denen man noch zurückkehren kann, blau sind die vollständig bearbeiteten Orte. Zurück in Ulm ist nun Stuttgart an der Reihe, da Augsburg und Lindau schon besucht wurden.



Gesamtablauf der Tour:
M → A → UL → LI; LI fertig; zurück nach UL; UL → S; S fertig; zurück nach UL; UL → WÜ → FD; FD fertig; zurück nach WÜ; (Bild) WÜ → HO → N → R; R fertig; zurück nach N; N fertig; zurück nach HO; HO fertig; zurück nach WÜ; WÜ fertig; zurück nach UL; UL fertig; zurück nach A; A fertig; zurück nach M; M → RO; RO fertig; zurück nach M; M fertig – geschafft, alle Knoten des Graphen wurden besucht!

Die Tiefensuche

Da man bei dieser Strategie immer erst entlang eines Pfads so weit in die Tiefe des Graphen vordringt, bis es nicht mehr weitergeht, nennt man diesen Algorithmus **Tiefensuche**. Er beinhaltet ein typisch rekursives Backtracking-Vorgehen:

- dasselbe Verfahren auf veränderte, einfachere Situationen anwenden,
- im Ablauf zurückgehen, wenn kein weiterer Schritt mehr möglich ist,
- Abbruch, wenn das gewünschte Ziel erreicht ist.

Im Laufe der Tiefensuche nimmt ein Knoten zwei Zustände ein: „unbesucht“ und „besucht“. Der Algorithmus wird beendet, wenn alle Rekursionsstufen abgearbeitet sind. Bei einem zusammenhängenden Graphen sind dann auch alle Knoten besucht.

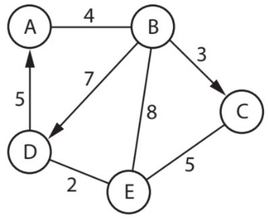
```
methode KnotenBesuchen(aktuell: GANZZAHL)
    besuchteKnoten.Hinzufügen(aktuell)
    zähle nummer von 0 bis knoten.ElementanzahlGeben() - 1
        falls (matrix.ElementGeben(aktuell).ElementGeben(nummer) > 0) und
              (NICHT besuchteKnoten.Enthält(nummer))
        dann
              KnotenBesuchen(nummer)
        endefalls
    endezähle
endemethode
```

Die Adjazenzmatrix wird nach Verbindungen zu Knoten durchsucht, die noch nicht besucht wurden.

Mit dem Algorithmus **Tiefensuche** können alle Knoten eines zusammenhängenden Graphen systematisch besucht werden. Bei diesem Algorithmus dringt man immer erst so weit wie möglich in die Tiefe des Graphen vor, bevor man schrittweise wieder zurückgeht, um die restlichen Abzweigungen der Vorgängerknoten abzuarbeiten.
Die Tiefensuche ist ein Backtracking-Algorithmus.

Aufgaben

1 Graphen – startklar



a Erläutern Sie anhand des rechts stehenden Graphen die Begriffspaare gewichtet/ungewichtet, gerichtet/ungerichtet, zyklisch/zyklenfrei und zusammenhängend/nicht zusammenhängend und geben Sie dabei an, welche der Begriffe auf diesen Graphen zutreffen.

b Erläutern Sie den Begriff Adjazenzmatrix und stellen Sie die Adjazenzmatrix für den gegebenen Graphen auf.

2 Tiefensuche auf Papier

Schreiben Sie für den Graphen aus dem Lehrtext die Reihenfolge der Knoten auf, wie sie beim Algorithmus Tiefensuche durchlaufen werden. Ausgangspunkt sei WÜ (Würzburg). Notieren Sie die Knoten mit ihren Bezeichnern und die Kanten durch „Startknoten/Zielknoten".

3 Tiefensuche als Rollenspiel

Ca. 7 – 8 Personen stellen die Knoten eines Graphen dar. Die Kanten werden durch Schnüre dargestellt. Jeder Knoten erhält eine Karte mit dem Knotennamen; auf der einen Seite der Karte steht zusätzlich „besucht", auf der anderen „unbesucht". Eine Person übernimmt die Rolle des aktuellen Knotens; diese Person wird von jedem Knoten in alphabetischer Reihenfolge zu seinen noch nicht besuchten Nachbarknoten geschickt. Jeder Knoten ist selbst dafür verantwortlich, den aktuellen Knoten in der richtigen Reihenfolge weiter zu schicken, bzw. zum Vorgängerknoten zurückzuschicken.

4 Tiefensuche implementieren

Das gegebene Projekt enthält bereits Beispielgraphen für verschiedene Anwendungen.

a Ergänzen Sie den Rumpf der Methode *KnotenBesuchen* entsprechend des im Lehrtext gezeigten Pseudocodes. Testen Sie mit dem Beispielgraph „Autobahn".

b Ergänzen Sie die Methode *KnotenBesuchen* um eine Textausgabe, die den jeweils besuchten Knoten anzeigt. Ergänzen Sie eine zweite Textausgabe, die nach jeder Rückkehr aus der Rekursion „Zurück nach" sowie den Knotennamen ausgibt. Notieren Sie zunächst von einem beliebigen Startknoten aus die Reihenfolge, in der Sie die Knotenbesuche erwarten. Überprüfen Sie dann, ob die Ausgabereihenfolge der von Ihnen erwarteten Reihenfolge entspricht.

c Testen Sie nun mit der Methode für „Beispielgraph2" und Startknoten „A". Geben Sie an, wie sich das Ergebnis von den bisher verwendeten Graphen unterscheidet, und begründen Sie den Unterschied. Geben Sie einen Startknoten an, bei dem ein anderes Ergebnis entsteht.

d Für Schnelle: Ergänzen Sie in der Klasse GRAPH_MATRIX eine Methode *Zusammenhang-Testen*, welche die Methode *KnotenBesuchen* verwendet und anschließend ausgibt, ob der Graph zusammenhängend ist oder nicht.
Tipp: Nutzen Sie die Länge des Feldes der besuchten Knoten.

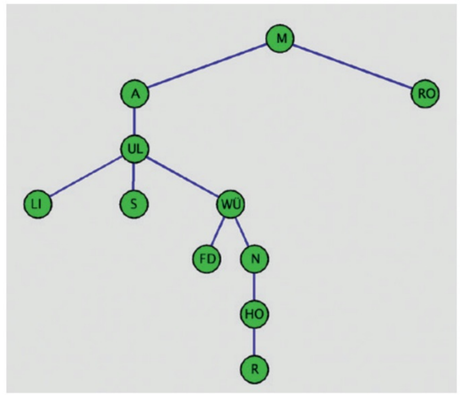
5 Tiefensuche anders implementieren

Anstatt bei der Tiefensuche ein Feld mit den bereits besuchten Knoten anzulegen, ist es auch möglich, ein Attribut besucht: WAHRHEITSWERT bei der Klasse KNOTEN zu ergänzen.

a Ändern Sie Ihre Lösung aus Aufgabe 4c) so ab, dass die Information, ob ein Knoten bereits besucht wurde, nun als Attribut in der Klasse KNOTEN gespeichert ist. Testen Sie Ihre neue Implementierung mit den in der Aufgabe gegebenen Graphen.

b Vergleichen Sie die beiden Lösungen und geben Sie begründet an, welche Variante Sie vorziehen würden.

6 Tiefensuchbaum

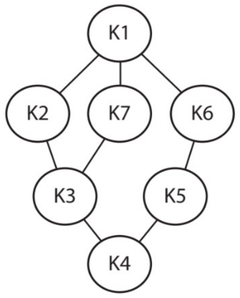


Da als Ergebnis der Tiefensuche jeder Knoten – mit Ausnahme des Startknotens – genau einen Vorgänger zugeteilt bekommt, lässt sich durch ein Baumdiagramm sehr schnell ein Überblick über alle Pfade von einem Startknoten aus erhalten. Das Bild rechts zeigt den Baum, der für das Beispiel aus dem Lehrtext bei der Suche von „M" aus entsteht.

a Erstellen Sie für den Graphen aus dem Lehrtext den Tiefensuchbaum für die Suche von Knoten „UL" aus.

b Ergänzen Sie in dem Vorlagenprojekt die Methoden *KnotenBesuchen* und *TiefensucheAusführen* so, dass bei der Tiefensuche auch der Tiefensuchbaum erstellt und anschließend ausgegeben wird. Die Klasse KNOTEN stellt dazu eine Methode *NeuesBlattAnfügen(blatt: KNOTEN)* zur Verfügung. Zur Visualisierung können Sie die *Methode BaumAnzeigen(wurzel: KNOTEN)* der Klasse GRAPH_MATRIX verwenden.
Testen Sie insbesondere mit dem Graphen aus dem Lehrtext und vergleichen Sie den von Ihnen in Teilaufgabe a) erstellten Tiefensuchbaum mit dem Ergebnis des Programms.

7 Tiefensuche versus Breitensuche



Sowohl die Tiefen- als auch Breitensuche dienen dazu, alle Knoten eines Graphen zu durchlaufen. Die beiden Algorithmen haben aber deutliche Unterschiede.

a Geben Sie für beide Algorithmen die Reihenfolge der besuchten Knoten für den Startknoten K1 bei Abarbeitung in alphabetischer Reihenfolge an.

b Begründen Sie für beide Algorithmen am Beispiel rechts, dass nicht alle Kanten durchlaufen werden.

c Geben Sie an, wie ein Graph aussehen muss, dass bei der Tiefensuche auch alle Kanten besucht werden. Geben Sie weiter an, ob bei diesem Graphen auch bei der Breitensuche alle Kanten durchlaufen werden.

d Erläutern Sie die konzeptionellen Unterschiede der beiden Algorithmen. Gehen Sie auch darauf ein, welche zusätzliche Datenstruktur jeweils benötigt wird.

***8 Tiefensuche mit Adjazenzlisten**

Gibt es in einem Graphen viele Knoten, aber nur relativ wenige Kanten, so äußert sich das in Form von vielen „leeren" Zellen in der Adjazenzmatrix. Bei der Speicherung des Graphen mit Hilfe einer Adjazenzmatrix wird in diesen Fällen sehr viel Speicherplatz verschwendet.
Eine Möglichkeit, mit dieser Situation speichersparend umzugehen, ist die Verwendung sogenannter Adjazenzlisten. Hier wird bei jedem Knoten ein Feld mit den von diesem Knoten ausgehenden Kanten gespeichert; als Kantenattribute sind in diesem Fall nur das Gewicht der Kanten und eine Referenz auf deren Zielknoten nötig.

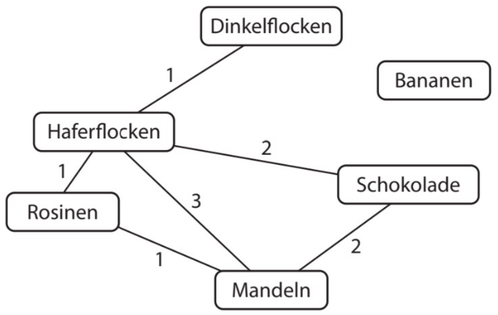
a In den Klassen KNOTEN, KANTE und GRAPH_LISTE sind bereits die grundlegenden Attribute und Methoden zum Einlesen und Aufbauen des Graphen vorhanden. Vergleichen Sie diese Implementierung mit der bisher verwendeten Implementierung in GRAPH_MATRIX. Geben Sie an, inwieweit vergleichbare Methoden einfacher oder komplexer geworden sind.

b Implementieren Sie nun den Rumpf der Methode *KnotenBesuchen*. Testen Sie mit den vorhandenen Graphen.

c Vergleichen Sie die neue Implementierung der Methode *KnotenBesuchen* mit der bisherigen Implementierung. Geben Sie begründet an, welche Implementierung Sie für besser halten.

9 Die Mischung machts (aus Abitur 2016)

Während der Zusammenstellung seiner Müslimischung sollen einem Kunden weitere Zutaten zu seiner aktuell ausgewählten Zutat vorgeschlagen werden. Hat der Kunde z. B. als aktuelle Zutat Haferflocken ausgewählt, sollen ihm die Zutaten empfohlen werden, die bereits andere Kunden mit Haferflocken zusammengestellt haben.
Um diese Empfehlungen geben zu können, müssen die Daten über die Zusammenstellungen gespeichert werden.
Folgende vier Zutatenkombinationen wurden im abgebildeten Graphen ausgewertet.
Kombination A: Haferflocken, Rosinen und Mandeln
Kombination B: Haferflocken, Mandeln und Schokolade
Kombination C: Dinkelflocken und Haferflocken
Kombination D: Haferflocken, Schokolade und Mandeln



Die Zahl an jeder Kante gibt an, in wie vielen Kombinationen beide Zutaten enthalten sind. Ist keine Kante vorhanden, trat diese Zutatenkombination noch nicht auf.

a Nennen Sie zwei wesentliche Eigenschaften dieses Graphen. Repräsentieren Sie den gegebenen Graphen durch eine Adjazenzmatrix.

In der Klasse MUESLIMATRIX soll der Graph auf der Basis einer Adjazenzmatrix implementiert sein. Die Bezeichner der Knoten sollen ebenfalls in dieser Klasse in einem Feld von Zeichenketten gespeichert werden.

b Geben Sie für die Klasse MUESLIMATRIX […] die Deklaration der Attribute sowie eine Implementierung des Konstruktors an. Im Konstruktor werden den Knoten die Zutatenbezeichner Bananen, Dinkelflocken, Haferflocken, Mandeln, Rosinen und Schokolade zugeordnet; ferner wird eine Adjazenzmatrix erstellt und deren Einträge explizit mit der Zahl 0 besetzt.

c Die Methode *dazuEmpfehlen(i)* soll aus der Adjazenzmatrix auslesen, welche Zutaten bereits mit der aktuellen Zutat, die im Knotenfeld den Index i hat, mehr als einmal kombiniert wurden und diese dem Kunden als Empfehlung anzeigen. Für den obigen Graphen soll für die Zutat „Mandeln" beispielhaft folgendes ausgegeben werden:
„Viele Kunden haben zu Ihrer Zutat noch Haferflocken gewählt. Viele Kunden haben zu Ihrer Zutat noch Schokolade gewählt."
Stellen Sie den Algorithmus für die Umsetzung der Methode *dazuEmpfehlen(i)* graphisch dar, z. B. durch ein Struktogramm.

d Für die Entwicklung neuer Müslimischungen ist es hilfreich zu wissen, welche Zutaten auch indirekt über andere Zutaten miteinander in Beziehung stehen (im obigen Graphen z. B. Dinkelflocken und Rosinen). Formulieren Sie unter Bezug auf die Adjazenzmatrix einen Algorithmus, mit dem ausgehend von einem Startknoten alle erreichbaren Knoten genau einmal ausgegeben werden.

10 Firmenbeteiligungen (aus Abitur 2014)

Viele Firmen sind an anderen Firmen beteiligt. Durch Fusionen, Kooperationen oder Veräußerungen kann es vorkommen, dass Firmen auch gegenseitig unterschiedlich große Firmenanteile voneinander besitzen.
Firma A besitzt 49 Prozent der Anteile von Firma B und 12 Prozent der Anteile von Firma C. Letztere besitzt wiederum 3 Prozent von Firma A. Die Firmen E und F sind jeweils zu 25 Prozent an Firma G beteiligt, die wiederum Anteile in Höhe von 84 Prozent an Firma C hält. Firma F besitzt außerdem 10 Prozent der Firmenanteile von Firma D, umgekehrt hält Firma D einen Anteil von 33 Prozent von Firma F. Die Firmen E und B haben sich im Zug einer engen Kooperation gegenseitig jeweils 7 Prozent der Firmenanteile überschrieben.

a Stellen Sie die dargelegte Situation mithilfe eines geeigneten Graphen dar. Begründen Sie, dass hier ein ungerichteter Graph ungeeignet ist.

b Geben Sie die zum Graphen von Teilaufgabe a) gehörige Adjazenzmatrix an.

c Bei der Implementierung wird die Adjazenzmatrix von Teilaufgabe b) in einem zweidimensionalen Feld adjazenzmatrix implementiert; das Attribut anzahlKnoten steht für die Knotenanzahl.
Formulieren Sie unter Verwendung dieser Attribute einen Algorithmus zum Graphendurchlauf. Geben Sie anschließend die Reihenfolge der besuchten Knoten an, wenn Sie Ihren Algorithmus auf den Graphen aus Teilaufgabe a) mit A als Startknoten anwenden. Geben Sie zudem die Bedeutung der Menge der besuchten Knoten im beschriebenen Anwendungsfall an.

3.4 Anwendungen der Tiefensuche: Optimale Lösungen

Das gegebene Projekt sucht den Weg von einem Ort zu einem anderen.

a Erzeugen Sie ein Objekt der Klasse WEGE und rufen Sie die Methode *ReiseStarten1* mit Startknoten „M“ und Zielknoten „N“ auf. Wenn ein Weg gefunden ist, hält das Programm an und macht nach Tastendruck weiter. Lassen Sie sich alle gefundenen Wege anzeigen und notieren Sie dabei, welcher Weg optimal wäre und warum.

b Wiederholen Sie Aufgabe a) mit einem anderen Paar von Start- und Zielknoten.

c Stellen Sie eine Vermutung auf, wie der Algorithmus der Tiefensuche modifiziert worden sein könnte, damit er alle möglichen Wege anzeigt und nicht mehr nur alle Knoten besucht. Nennen Sie auch eine mögliche Abbruchbedingung der Rekursion.
Öffnen Sie den Quellcode der Methode *KnotenBesuchen1* in der Klasse GRAPH_MATRIX und vergleichen Sie Ihre Vermutung mit der Implementierung.

d Skizzieren Sie eine Strategie, um aus allen Lösungen des Algorithmus von *KnotenBesuchen1* den optimalen Weg herauszufinden und am Ende zu zeigen.

e Verwenden Sie nun die Methode *ReiseStarten2*, lassen Sie sich den Weg für die Knotenpaare aus a) bzw. b) anzeigen und verifizieren Sie, dass diese Wege wirklich optimal sind.

f Betrachten Sie nun den Quellcode von *KnotenBesuchen2* der Klasse GRAPH_MATRIX. Geben Sie an, nach welcher Strategie die optimale Lösung gefunden wurde und ob diese Strategie mit Ihrer Strategie aus Aufgabe c) übereinstimmt.

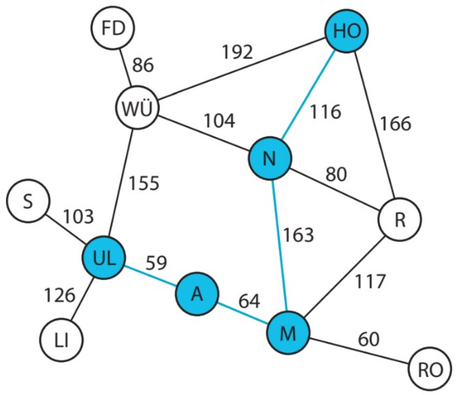
g Für Schnelle: Ändern Sie die Reihenfolge beim Untersuchen der Knoten (z. B. von hinten her). Geben Sie an, ob und wenn ja wie diese Änderung die jeweils gefundene Lösung beeinflusst.

Verschiedene Ziele

Außer der Grundaufgabe, alle Knoten zu besuchen, kann die Tiefensuche auch zur Suche nach einem ganz bestimmten Knoten oder nach Knoten mit einer vorgegebenen Eigenschaft genutzt werden. Da die Tiefensuche ein echtes Backtracking-Verfahren ist, kann sie auch alle Wege zum Ziel suchen und damit den optimalen Weg finden.

Einen bestimmten Knoten finden

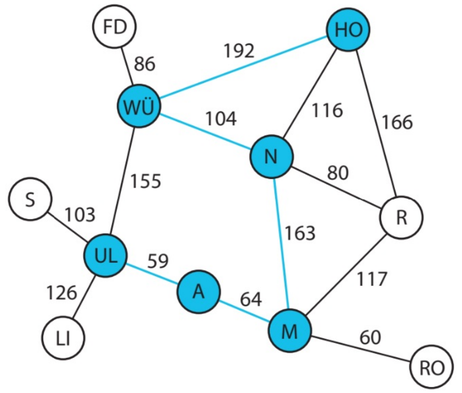
Wenn man in dem rechtsstehenden Graphen testen will, ob es einen Weg von Ulm nach Hof gibt, findet die Tiefensuche zuerst den Weg UL → A → M → N → HO (wegen der Regel, die Knoten immer in alphabetischer Reihenfolge zu probieren). Der zuerst gefundene Weg ist im Allgemeinen nicht der kürzeste Weg. Aber mit diesem Weg weiß man, dass der gesuchte Knoten erreichbar ist; wenn man nur daran interessiert ist, kann man die Suche an dieser Stelle beenden.



Den kürzesten Weg finden

In der Regel möchte man aber nicht irgendeinen Weg kennen; gesucht ist der kürzeste Weg, hier von Ulm nach Hof. Dazu muss man alle Wege zum Ziel gehen; der kürzeste Weg ist gesichert dabei. Hat man den Zielknoten Hof erst einmal erreicht, braucht man dabei nicht weiter zu

suchen; jeder dann noch gefundene Weg wäre auf jeden Fall länger, weil er erst wieder von Hof weg und dann wieder hinführen würde. Die Rekursion terminiert also an dieser Stelle wie beim Erreichen einer Sackgasse; das Backtracking führt zurück nach Nürnberg.
Im nächsten Schritt würde von Nürnberg aus der Weg über Regensburg versucht. In der bisherigen Implementierung würde von Regensburg aus Hof aber nicht mehr betreten werden, da es ja schon besucht wurde. Daher muss der Knoten für Hof bei der Rückkehr aus dem Rekursionsschritt wieder als „noch nicht besucht“ markiert werden, um auf einem anderen Weg neu betreten werden zu können.
Als nächstes würde die Wegalternative von Nürnberg aus über Würzburg nach Hof gefunden werden. Hier sieht man: Auch der Knoten für Würzburg muss danach wieder den Status „noch nicht besucht“ erhalten, wenn er vollständig bearbeitet wurde, sonst könnte später der Weg UL → WÜ → HO nicht mehr gefunden werden.
Zusammengefasst müssen also zwei Veränderungen am Algorithmus der Tiefensuche vorgenommen werden, um alle Lösungen der Aufgabe „Bestimme Pfad von A nach B" zu finden:



- Der letzte Rekursionsschritt terminiert, wenn der gesuchte Knoten erreicht ist.
- Ist ein Knoten in einem Rekursionsschritt vollständig abgearbeitet, muss er wieder auf den Status „nicht besucht“ gesetzt werden.

Besten Weg speichern

Wenn man den optimalen Pfad von A nach B sucht, muss man den optimalen Pfad, den aktuellen Pfad sowie jeweils das Gütekriterium (im Beispiel die Weglänge, allgemein die Summe der Kantengewichte auf dem Pfad) in geeigneten Attributen speichern. Hat man einen neuen Pfad gefunden, vergleicht man, ob dieser „besser“ ist, und ersetzt dann gegebenenfalls das bisherige Optimum durch die jetzt gefundene Lösung.

Es kann auch mehrere geeignete Ziele geben

Mit dieser Strategie werden nicht nur alle Pfade von A nach B gefunden, es können auch alle Pfade von einem Knoten A zu einem Knoten mit gegebener Eigenschaft gefunden werden, auch wenn es mehrere geeignete Knoten gibt. So kann man alle Pfade vom Standort zu einem Knoten mit Tankstelle im Umkreis suchen lassen und dann den Weg zur nächsten Tankstelle ausgeben.

In leicht abgewandelter Version kann die Tiefensuche verwendet werden, um zu prüfen, ob ein Knoten mit einer bestimmten Eigenschaft erreichbar ist.
Eine weitere Modifikation erlaubt, alle Pfade zu Knoten mit einer bestimmten Eigenschaft zu suchen und so den optimalen Pfad zu einem Knoten mit einer bestimmten Eigenschaft zu ermitteln.
Eine typische Aufgabe dafür ist die Suche des kürzesten Weges von A nach B in einem Graphen, der die Entfernungsangaben zwischen den Orten als Kantengewichte enthält.

Aufgaben

1 Alle Wege auf Papier
Notieren Sie für den Autobahngraphen aus dem Lehrtext alle Pfade für den Weg von Ulm („UL") nach Hof („HO") in der Reihenfolge, wie sie sich beim dargestellten Algorithmus ergeben. Verwenden Sie für den Pfad die Schreibweise wie im Lehrtext und notieren Sie bei jedem Pfad die Summe der Kantengewichte (die Weglänge). Tipp: Es gibt 10 Pfade.

2 Knoten mit einer bestimmten Eigenschaft suchen (Teil 1)
Eine Vertreterin hat in ihrer Landkarte alle Orte markiert, in denen sie mindestens eine Kundin oder einen Kunden hat. Weiter sind die Orte markiert, in denen ein Geschäft ihrer Firma ist.
Im gegebenen Projekt sind bei den Knoten zusätzlich die Attribute kundeVorhanden und ladenVorhanden (WAHRHEITSWERT) ergänzt sowie die Methoden *IstKundeVorhanden()* bzw. *IstLadenVorhanden()*, die jeweils den Wert des Attributs zur Verfügung stellt.

a Ändern Sie die Methoden *TiefensucheAusführen* und *KnotenBesuchen* der Klasse GRAPH_MATRIX so ab, dass die Suche beendet wird, wenn ein Knoten mit einem Laden gefunden wurde. Markieren Sie auch die in Arbeit befindlichen Knoten, so dass der Weg grob angezeigt wird.
Tipp: *KnotenBesuchen* benötigt als Resultat einen Wahrheitswert.

b Testen Sie Ihre Lösung mit der Suche von verschiedenen Startpunkten aus. Begründen Sie, warum nicht immer der kürzeste Weg gefunden wird.

c Für Schnelle: Ergänzen Sie die Methoden *TiefensucheAusführen* und *KnotenBesuchen* so, dass als Parameter angegeben werden kann, nach welchem der vorhandenen Kriterien gesucht werden soll.

3 Labyrinth (Teil 1)
Das Tiefensuche-Verfahren hilft, systematisch einen Weg aus einem Labyrinth zu finden. Alle Kreuzungen und Verzweigungen werden als Knoten betrachtet, ebenso die Enden der Sackgassen. Die Gänge sind die Kanten des Graphen.

a Der griechischen Mythologie zufolge fand der Held Theseus mithilfe eines langen Fadens, den ihm die Prinzessin Ariadne geschenkt hatte (Ariadnefaden), den Weg aus dem Labyrinth, in dem sich das Monster Minotauros befand. Beschreiben Sie, wie man, unter Verwendung eines langen, auf- und abspulbaren Fadens, ein Labyrinth nach dem Tiefensuchverfahren ablaufen kann, bis man den Ausgang erreicht hat. Nennen Sie eine mögliche Regel, wie man an einer Verzweigungsstelle die möglichen Gänge für das Weitergehen auswählt. Geben Sie auch an, woran Sie erkennen können, dass ein Knoten (eine Verzweigungsstelle) schon besucht wurde.

b Ergänzen Sie den Rumpf der Methode *KnotenBesuchen* in Klasse GRAPH_MATRIX gemäß der Strategie in Aufgabe 2. Fügen Sie zusätzlich beim Eintritt in die Methode den aktuellen Knoten an das Feld weg an; bei einer erfolglosen Suche (Rückkehr mit FALSCH) müssen Sie den Knoten wieder vom Weg entfernen. Starten Sie die „Flucht" nach den Anweisungen im Projekt.

4 Den kürzesten Weg zum Ziel suchen

a Ergänzen Sie in der Klasse GRAPH_MATRIX die Attribute aktuellerWeg, optimalerWeg und optimaleLänge so, dass Sie dort den aktuell von der Tiefensuche gefundenen Weg sowie den optimalen Weg und dessen Länge speichern können.

b Ergänzen Sie den Rumpf der Methode *KnotenBesuchen* der Klasse GRAPH_MATRIX so, dass bei der Tiefensuche alle Wege zum gegebenen Zielknoten gefunden werden, der aktuelle Weg gespeichert wird und gegebenenfalls beim Erreichen des Ziels ein neuer, besserer Weg gespeichert wird.

c Ergänzen Sie den Rumpf der Methode *TiefensucheAusführen* der Klasse GRAPH_MATRIX so, dass nach der Tiefensuche der optimale Weg im Graphen angezeigt wird.

d Für Schnelle: Vergleichen Sie mit der Strategie des Dijkstra-Algorithmus (→ 11. Klasse, bei Bedarf Internetrecherche).

5 Knoten mit einer bestimmten Eigenschaft suchen (Teil 2)

a Ändern Sie Ihren Algorithmus aus Aufgabe 2b) so ab, dass nun nicht der Weg zu irgendeinem Knoten mit einer Kundin oder einem Kunden gefunden wird, sondern der Weg zu dem Knoten mit der geringsten Entfernung. Testen Sie Ihre Lösung mit der Suche von verschiedenen Startpunkten aus.

b Für Schnelle: Ergänzen Sie auch hier die Methoden *TiefensucheAusführen* und *KnotenBesuchen* so, dass als Parameter angegeben werden kann, nach welchem der vorhandenen Kriterien gesucht werden soll.

6 Labyrinth (Teil 2)
Ergänzen Sie Ihre Lösung aus Aufgabe 3 gemäß der Strategie im Lehrtext (bzw. Aufgabe 4) so, dass der kürzeste Weg zum Ausgang gefunden und angezeigt wird.

7 Der Handelsreisende („traveling salesman")
Eine weitere „klassische" Backtracking-Aufgabe neben dem Rucksackproblem ist das Problem des Handelsreisenden. Von seinem Büro aufbrechend muss er alle seine Kundinnen und Kunden besuchen. Anschließend muss er wieder zurück in das Büro. Der Weg soll möglichst kurz sein. Dabei ist es nicht zu vermeiden, dass manche Orte doppelt (oder öfter) besucht werden. Um diese sehr lästige Komplikation vom Programm fernzuhalten, trägt man Wege zwischen allen Orten ein. Für den Algorithmus ist nur wichtig, dass z. B. die Entfernung von München nach Würzburg 167 km ist. Ob dieser Weg über Nürnberg oder Ulm führt, ist für das Ergebnis (und für den Vertreter) nicht wesentlich.

a Implementieren Sie die Methoden *TiefensucheAusführen* und *KnotenBesuchen* so, dass zu dem gegebenen Wegenetz und einem angebbaren Startknoten (Ort für das Büro) der optimale Routenplan bestimmt wird.

b Für Schnelle: Wenn ein Vertreter seinen Bezirk neu aufbaut, kann er sich den Ort für das Büro wählen. Erweitern Sie Ihre Lösung von Teilaufgabe a) so, dass der beste Ort für das Büro berechnet wird.

***8 Wegesuche mit Adjazenzlisten**
Ergänzen Sie die Klasse GRAPH_LISTE so, dass mit Hilfe der Tiefensuche der kürzeste Weg von einem als Parameter gegebenen Startort zu einem ebenfalls als Parameter gegebenen Zielort gefunden wird. Testen Sie mit verschiedenen Start- und Zielorten.
Vergleichen Sie die relevanten Methoden mit der Lösung in Aufgabe 4 und geben Sie begründet an, welche Lösung Sie vorziehen würden.

9 Forschungsauftrag: Rechenzeitaufwand für Backtrackingalgorithmen
Schätzen Sie für kleine Graphen ab, wie viele Wege zur Ermittlung des kürzesten Weges betrachtet werden müssen, und begründen Sie damit, dass dieses Verfahren für große Graphen so nicht geeignet ist.
Recherchieren Sie allgemein zum Rechenzeitaufwand von Backtrackingverfahren.

Teste dich selbst

T1 Richtig oder falsch?
Beurteilen Sie, ob folgende Aussagen richtig oder falsch sind. Begründen Sie Ihre Meinung bei falschen Aussagen und geben Sie eine berichtigte Aussage an:

a Rekursive Methoden rufen sich selbst auf.
b Eine rekursive Methode braucht nicht notwendigerweise eine Abbruchbedingung.
c Rekursive Aufrufe sind immer linear.
d Beim Backtracking-Verfahren werden alle Lösungen eines Problems gefunden.
e Die Tiefensuche besucht alle Knoten eines zusammenhängenden Graphen.
f Bei der Tiefensuche werden die benachbarten Knoten des Startknotens zuerst besucht.
g Die Tiefensuche kann nicht zur Suche des kürzesten Wegs verwendet werden.

T2 Ich check's, dank deiner Hilfe!
Sie wollen einer Mitschülerin bei der Vorbereitung auf die Informatik-Schulaufgabe helfen. Erklären Sie ihr dazu folgende Begriffe anhand geeigneter Beispiele: Rekursion, rekursiver Aufruf, Abbruchbedingung, Rekursionsende, lineare Rekursion, verzweigte Rekursion, Backtracking, optimale Lösung beim Backtracking, Tiefensuche sowie kürzester Weg.

T3 Programmieren (aus Abitur 2017)
Music17 wertet das Hörverhalten seiner Kunden aus und erzeugt daraus Listen, die unter der Rubrik „Andere hörten auch" bei jedem Song abgerufen werden können. Wenn ein Hörer von einem Titel zu einem weiteren wechselt, wird dies registriert.
Die Anzahl der erfolgten Wechsel wird in einer Adjazenzmatrix gespeichert. Die Songtitel werden in der Matrix zur Vereinfachung […] durch ihre Anfangsbuchstaben repräsentiert:

	A	B	C	G	H	I	L
A		15	25				
B	10		2	12			
C	6			5			
G						30	
H				10			
I				30			3
L				4			

Hinweis: Hier wurde z. B. zweimal von Song B zu Song C gewechselt.

a Begründen Sie anhand der Adjazenzmatrix, warum der zugehörige Graph gerichtet ist, und zeichnen Sie diesen.
b Für den Song „Am Ende wird alles gut" sollen alle Titel ermittelt werden, die von ihm aus über eine oder mehrere Kanten, deren Kantengewicht jeweils mindestens 10 beträgt, erreichbar sind.
Formulieren Sie einen rekursiven Algorithmus, der dies leistet.
Geben Sie für den Startknoten A die ermittelten Titel in der durch Ihren Algorithmus festgelegten Reihenfolge an.

Zusammenfassung

Rekursion

Ein Verfahren heißt **rekursiv**, wenn es sich selbst auf immer einfachere Teile der Anfangssituation anwendet, bis die Situation am Ende so einfach ist, dass sie direkt gelöst werden kann.
Rekursive Methoden rufen sich selbst direkt oder indirekt so lange auf (Rekursionsschritt), bis eine **Abbruchbedingung** erreicht ist, so dass die Selbstaufrufe zu einem Ende kommen (Rekursionsende).
Je nach Anzahl der Selbstaufrufe spricht man von **linearer** (ein Aufruf) oder **verzweigter** (2 oder mehr Aufrufe) **Rekursion**.

Lineare Rekursion

```
methode SchrittAusführen
  (anzahl: GANZZAHL)
    falls anzahl == 1
    dann
        // Anweisungen
    sonst
        SchrittAusführen(anzahl - 1)
        // Anweisungen
    endefalls
endemethode
```

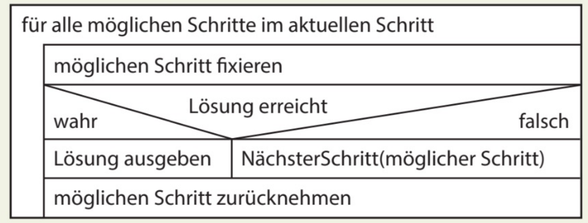
Verzweigte Rekursion

```
methode SchrittAusführen
  (stufe: GANZZAHL)
    falls stufe == 1
    dann
        // Anweisungen
        Drehen(drehtLinks)
        turtle.Gehen(50)
    sonst
        SchrittAusführen(stufe - 1)
        // Anweisungen
        SchrittAusführen(stufe - 1)
    endefalls
endemethode
```

Backtracking

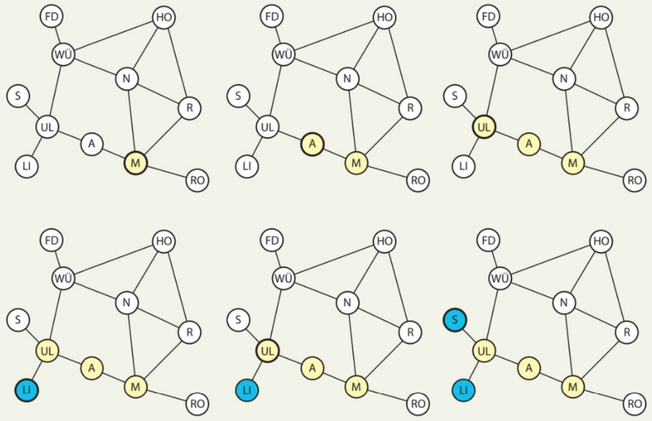
Mit Hilfe des **Backtracking-Verfahrens** können systematisch **alle Lösungen** einer gegebenen Situation rekursiv gesucht werden.
Damit kann auch die **optimale Lösung** gefunden werden.

NächsterSchritt(aktueller Schritt)



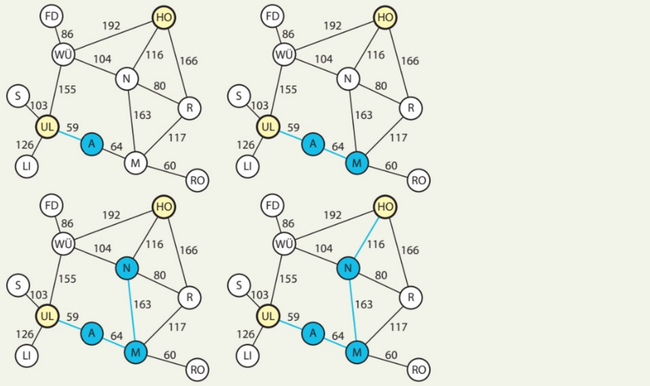
Tiefensuche

Mit dem Algorithmus **Tiefensuche** können alle Knoten eines zusammenhängenden Graphen systematisch besucht werden. Bei diesem Algorithmus dringt man immer erst so weit wie möglich in die Tiefe des Graphen vor, bevor die restlichen Abzweigungen der Vorgängerknoten abgearbeitet werden.
Die Tiefensuche ist ein Backtracking-Algorithmus.

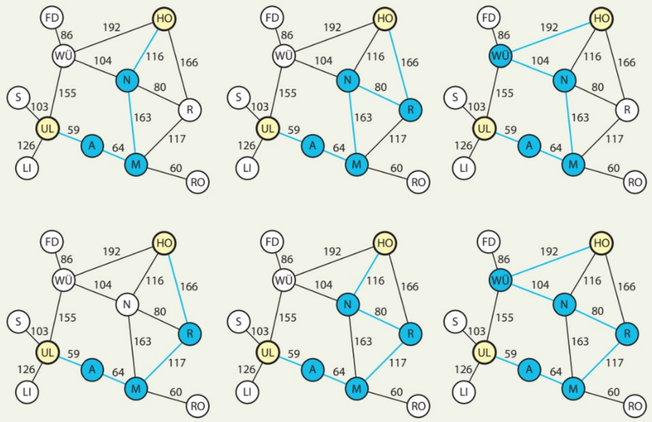


Tiefensuche: Optimale Lösungen

In leicht abgewandelter Version kann die Tiefensuche verwendet werden, um zu prüfen, ob ein Knoten mit einer bestimmten Eigenschaft erreichbar ist.



Eine weitere Abwandlung erlaubt, alle Pfade zu Knoten mit einer bestimmten Eigenschaft zu suchen und so den optimalen Pfad zu einem Knoten mit einer bestimmten Eigenschaft zu suchen.
Eine typische Aufgabe dafür ist die Suche des kürzesten Weges von A nach B in einem Graphen, der die Entfernungsangaben zwischen den Orten als Kantengewichte enthält.



Zum Weiterlesen

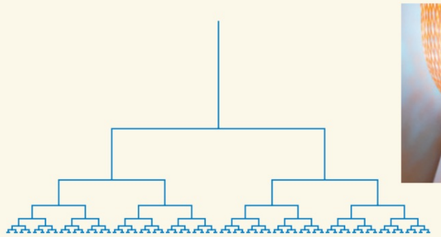
L4 Fraktale – Selbstähnlichkeit

Fraktal ist ein von Benoît Mandelbrot geprägter Begriff für geometrische Muster mit einem hohen Grad an Selbstähnlichkeit. Solche Muster werden nicht nur künstlich erzeugt (rechts), sondern kommen auch in der Natur vor, wie bei Küstenlinien, Farnen (unten) oder Adernetzen.

Mandelbrot selbst schreibt 1975 dazu:
„Wolken sind keine Kugeln, Berge keine Kegel, Küstenlinien keine Kreise. Die Rinde ist nicht glatt – und auch der Blitz bahnt sich seinen Weg nicht gerade. […]
Die Existenz solcher Formen fordert uns zum Studium dessen heraus, was Euklid als formlos beiseitelässt, führt uns zur →Morphologie des →Amorphen. Bisher sind die Mathematiker jedoch dieser Herausforderung ausgewichen. Durch die Entwicklung von Theorien, die keine Beziehung mehr zu sichtbaren Dingen aufweisen, haben sie sich von der Natur entfernt. Als Antwort darauf werden wir eine neue Geometrie der Natur entwickeln und ihren Nutzen auf verschiedenen Gebieten nachweisen. Diese neue Geometrie beschreibt viele der unregelmäßigen und zersplitterten Formen um uns herum – und zwar mit einer Familie von Figuren, die wir Fraktale nennen werden."
Eine geometrische Figur wird selbstähnlich genannt, wenn sie aus verkleinerten Kopien ihrer selbst besteht. Selbstähnliche Figuren wurden schon in der euklidischen Geometrie beispielsweise durch Helge von Koch und Waćlaw Sierpiński untersucht.
Die Abbildung links unten zeigt eine der von diesen Mathematikern entwickelten Figuren, die sich geometrisch konstruieren lassen.
Auch ein Binärbaum ist eine selbstähnliche Figur. Die Faszination, die selbstähnliche Figuren ausüben, wird auch in Werbung und Kunst eingesetzt.

→Morphologie: griech. Gestalt, Form; Lehre von der Struktur und Form von Organismen.

→Amorphes Material: Stoffe, bei denen die Atome keine regelmäßige Anordnung einnehmen, sondern ein unregelmäßiges Muster ergeben.



L5 Rekursion und Kellerspeicher

Methoden rekursiv aufzurufen, scheiterte in den ersten Hochsprachen wie COBOL und FORTRAN daran, dass die Speicherplätze für Parameter und lokale Variablen bereits zur Übersetzungszeit fest vorgegeben werden mussten.

Um eine Methode rekursiv aufrufen zu können, müssen aber für jede „Instanz" dieser Methode die Parameter und lokalen Variablen ihre eigenen, nur für diese Instanz gültigen Speicherplätze zugewiesen bekommen können. Das bedeutet insbesondere, dass die Speicherplätze erst bei der Ausführung des Programms zugewiesen werden können, da erst während der Abarbeitung bekannt wird, wie tief die rekursive Verschachtelung tatsächlich ist.

Einen geeigneten Mechanismus erfand der deutsche Informatikpionier F. L. Bauer mit dem Prinzip des Kellerspeichers (im Englischen stack: Stapel); 1957 ließ er dieses Prinzip patentieren. Wie bei vielen guten Ideen ist die Idee einfach, wenn man weiß, wie es geht, aber man muss zuerst einmal darauf kommen.

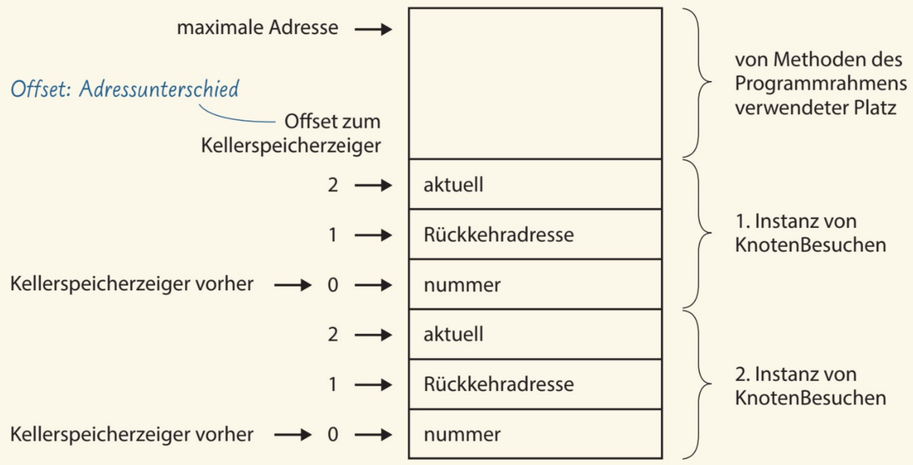
Laut einer Anekdote geht der Name Kellerspeicher darauf zurück, dass Prof. Bauer, als er das Konzept an der Tafel erläuterte und gefragt wurde, wie weit der Speicherplatz nach unten gehen würde, wenn es sehr viele rekursive Aufrufe wären, er antwortete: „Dann geht er eben in den Keller". Im Englischen hat sich der Begriff stack (Stapel) durchgesetzt, obwohl er bildlich eigentlich falsch ist, weil ein Stapel von unten nach oben wächst.

Die Adressen der Parameter und lokalen Variablen werden nicht absolut berechnet, sondern relativ zu einer globalen Variablen, dem Kellerspeicherzeiger. Der Kellerspeicherzeiger zeigt zu Beginn eines Programms auf die höchste adressierbare Speicherzelle. Jedes Mal, wenn eine Methode aufgerufen wird, wird der Kellerspeicherzeiger und der Platzbedarf für diese Methode erniedrigt, beim Beenden der Methode wird er wieder entsprechend erhöht.

Die Methode *KnotenBesuchen* bei der Tiefensuche benötigt einen Speicherplatz für den Parameter und einen weiteren Speicherplatz für eine lokale Variable.

```
methode KnotenBesuchen(aktuell: GANZZAHL)
    var nummer: GANZZAHL
    …
endemethode
```

Weiter muss auch noch die Rücksprungadresse (Adresse nach dem aufrufenden Befehl) gespeichert werden. Eine typische Struktur des Kellerspeichers sieht dann so aus.



Auf modernen Prozessoren wird der Kellerspeicher in der Regel durch ein eigenes Register für den Kellerspeicherzeiger unterstützt, genannt stack pointer (SP).

4 Nebenläufigkeit und Threads

In diesem Kapitel werden Sie …

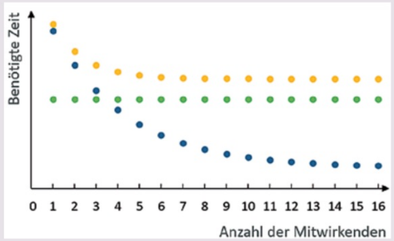
… eine große Aufgabe in mehrere parallele Teilaufgaben zerlegen.

… die Effizienz von Abläufen mit geeigneten Strategien steigern.

… Strategien zum Verhindern von Verklemmungen kennenlernen.

4.1 Nebenläufigkeit: Prozesse und Threads

Viele Aufgaben lassen sich allein oder im Team erledigen, wobei sich durch das gemeinsame Arbeiten sowohl Chancen als auch Herausforderungen ergeben können.



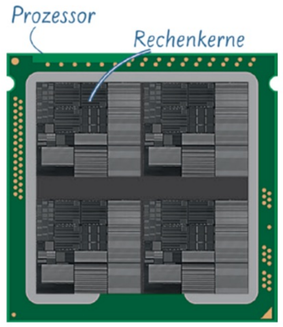
a Im Diagramm rechts wird für drei Aktivitäten jeweils skizziert, wie die für die Aktivität benötigte Zeit von der Teamgröße abhängt. Ordnen Sie die Verläufe je einer der folgenden Aktivitäten zu und begründen Sie Ihre Zuordnung kurz:
- i Backen eines Apfelkuchens
- ii Aufbacken einer Pizza
- iii Lösen von 100 Kopfrechenaufgaben

b Nennen Sie weitere Aktivitäten, welche dem grünen Kurvenverlauf im Diagramm zugeordnet werden können, und erläutern Sie jeweils, weshalb sich bei diesen Aktivitäten die benötigte Zeit durch eine Vergrößerung des Teams nicht verkürzt.

c Diskutieren Sie: Gibt es auch Aktivitäten, bei denen die benötigte Zeit mit steigender Teamgröße ansteigt? Was könnte hierfür die Ursache sein?

Parallelisierung ermöglicht Höchstleistungen

Die Vorteile von Parallelisierung begegnen uns in vielen Situationen des Alltags. So kann ein Gruppenprojekt in der Schule schneller abgeschlossen werden, wenn alle Mitglieder parallel an der Fertigstellung arbeiten, und Tageszeitungen können nur deshalb rechtzeitig erscheinen, weil viele Redakteurinnen und Redakteure gleichzeitig Artikel verfassen. Auch die enorme Rechenleistung vieler heutiger Computer kommt erst durch Parallelisierung zustande. Eine moderne CPU besteht dabei in der Regel aus mehreren weitgehend unabhängigen Rechenkernen (engl. „Cores“, siehe Abbildung rechts), die parallel Befehle abarbeiten können. Bei der Beschreibung von Computern oder Smartphones wird deshalb heutzutage oftmals die Anzahl der verbauten Rechenkerne als wichtiges Leistungsmerkmal hervorgehoben.

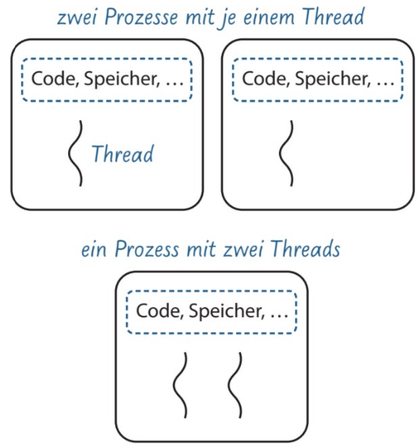


Nebenläufigkeit ermöglicht parallele Ausführung von Programmteilen

Viele Programmiersprachen sehen standardmäßig eine sequenzielle Abarbeitung des Programmcodes vor. Möchte man das Leistungspotential einer parallelen Abarbeitung nutzen, muss dies beim Programmieren explizit vorgegeben werden. Ein Programm mit mehreren Teileinheiten, die potentiell parallel abgearbeitet werden, also „nebeneinander ablaufen“ können, wird **nebenläufig** genannt. Dies bietet in verschiedenen Situationen Vorteile: Grafische Benutzeroberflächen werden z. B. oft nebenläufig in einem vom Rest des jeweiligen Programms unabhängigen Ausführungsstrang umgesetzt, damit diese auch dann flüssig bedienbar bleiben, wenn das Programm im Hintergrund andere Aufgaben ausführt. Rechenintensive Anwendungen sollten soweit möglich ebenso nebenläufig implementiert werden, damit sie bei der Ausführung möglichst alle Rechenkerne der CPU voll ausnutzen können. In der Praxis sind viele Problemstellungen allerdings nicht beliebig parallelisierbar. Zwei Berechnungen können z. B. nicht parallel ausgeführt werden, wenn das Ergebnis der ersten Rechnung für die zweite Berechnung benötigt wird. Auch der Zugriff auf bestimmte Ressourcen ist oft nicht parallel möglich; ein Drucker kann z. B. stets nur einen Druckauftrag gleichzeitig ausführen. Eine zentrale Herausforderung bei der Entwicklung nebenläufiger Programme ist es daher, die einzelnen Ausführungsstränge in geeigneter Weise aufeinander abzustimmen und, wenn erforderlich, auch aufeinander warten zu lassen.

Parallele Programmierung mit Threads

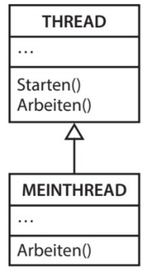
Startet man auf einem modernen Computer ein Programm, so wird vom Betriebssystem zunächst ein sogenannter **Prozess** eingerichtet, in dem die Ausführung des Programms und dessen Ressourcen verwaltet werden. Jedem Prozess wird dabei auch ein eigener Speicherbereich zugewiesen, in dem z. B. Variableninhalte und das Programm selbst abgelegt werden. Prozesse sind weitgehend unabhängig voneinander, was schnell zum Problem werden kann, wenn z. B. Zwischenergebnisse ausgetauscht werden sollen. Nebenläufigkeit kann daher auch innerhalb eines Prozesses über sogenannte →**Threads** realisiert werden. Ein Thread definiert dabei eine Ausführungsreihenfolge von Programmbefehlen. Standardmäßig existiert je Prozess zunächst nur ein Thread. Sofern durch das Programm keine weiteren Threads erzeugt werden, werden alle im Programmcode vorgegebenen Befehle sequenziell, d. h. nacheinander, abgearbeitet. Mittlerweile gibt es jedoch in vielen Programmiersprachen die Möglichkeit, zusätzliche Threads zu definieren, sodass das Programm in mehrere parallel ausführbare Einheiten aufgeteilt werden kann. Da diese Threads alle zum gleichen Prozess gehören, können sie den zum Prozess gehörenden Speicherbereich gemeinsam nutzen.



→ engl. „Faden“, im Deutschen manchmal auch „leichtgewichtiger Prozess“

Eigene Threads erzeugen

Je nach verwendeter Programmiersprache gibt es verschiedene Mechanismen, um mehrere Threads in einem Programm anzulegen und zu starten. Ein weit verbreitetes →Paradigma ist aber die Spezialisierung einer von der Programmierumgebung vorgegebenen Klasse THREAD. Die Ausführung eines auf diese Weise definierten Threads wird durch Aufrufen einer geerbten *Starten()*-Methode angestoßen. Im Thread wird daraufhin eine in der Klasse THREAD vorgegebene Methode (*Arbeiten* im Klassendiagramm) aufgerufen. Durch Überschreiben dieser Methode kann somit eigener Programmcode innerhalb des Threads ausgeführt werden. Nachdem ein Thread erzeugt und gestartet wurde, läuft er eigenständig, bis die Ausführung der überschriebenen Methode abgeschlossen ist. Die Zuteilung von Rechenzeit an die verschiedenen Threads übernimmt dabei in der Regel das Betriebssystem.



Java
```
class MeinThread extends Thread {
 @Override public void run() {
    …
  }
}
…
MeinThread t1 = new MeinThread();
t1.start();
```

Python
```
import threading

class MeinThread(threading.Thread):
    def run(self):
        …

t1 = MeinThread()
t1.start()
```

Swift
```
class MeinThread: Thread {
    override func main() {
        …
    }
}

let t1 = MeinThread()
t1.start()
```

→ grundsätzliche Denkweise/ Vorgehensweise

Nebenläufige Programme bestehen aus mehreren Ausführungssträngen, welche parallel ausgeführt werden können. Innerhalb eines Prozesses kann Nebenläufigkeit über **Threads** implementiert werden. Moderne Rechner arbeiten mit mehreren Rechenkernen, sodass mehrere Threads gleichzeitig ausgeführt werden können. Viele Algorithmen lassen sich jedoch aufgrund interner Abhängigkeiten nicht beliebig parallelisieren.

Aufgaben

1 Ein einfaches nebenläufiges Programm selbst programmieren

In dieser Aufgabe soll ein einfaches nebenläufiges Programm entwickelt werden, bei dessen Ausführung zwei Threads per Textausgabe miteinander „Ping Pong" spielen. Hierfür sollen zwei Unterklassen THREADPING und THREADPONG der Klasse THREAD angelegt werden. Wird ein Thread der Klasse THREADPING gestartet, soll dieser zehnmal hintereinander das Wort „Ping" auf der Konsole als Text ausgeben. Auf die gleiche Weise soll ein Thread der Klasse THREADPONG zehnmal das Wort „Pong" ausgeben.

- **a** Zeichnen Sie ein Klassendiagramm für das oben beschriebene Szenario.
- **b** Implementieren Sie nun das beschriebene Programm. Dabei soll nach dem Start des Programms je ein Thread der Klasse THREADPING und THREADPONG angelegt und gestartet werden.
- **c** Testen Sie Ihr Programm aus Teilaufgabe b) mehrfach und vergleichen Sie jeweils die Textausgabe. Beschreiben Sie Ihre Beobachtungen und geben Sie eine mögliche Erklärung für deren Zustandekommen an.

2 Effiziente Lagerarbeit

Starten Sie die Programmausführung, während derer zwei Lageristen nacheinander je eine Kiste an die rechte Wand schieben. Gestalten Sie das Programm dann unter Nutzung von Threads nebenläufig, so dass die beiden ihre Arbeit gleichzeitig erledigen können.

3 Mehr Threads = mehr Leistungsfähigkeit?

Mittels des bereitgestellten Programms kann die Leistungsfähigkeit eines Rechners untersucht werden. Das Programm bearbeitet hierfür 500 voneinander unabhängige aufwendige Berechnungen und misst die benötigte Zeit. Als Parameter kann beim Start des Programms festgelegt werden, mit wie vielen Threads das Programm arbeiten soll.

- **a** Lassen Sie das Programm mit 1, 2, 4, 8, 16, 32, 64, 128, 256 Threads arbeiten und notieren Sie sich die benötigte Zeit. Tragen Sie die Werte anschließend in ein Koordinatensystem ein (x-Achse: Threadanzahl, y-Achse: benötigte Zeit).
- **b** Interpretieren Sie den Werteverlauf im Koordinatensystem aus a) und stellen Sie basierend darauf eine Hypothese über die im Rechner verfügbare Anzahl an Rechenkernen auf.
- **c** Überprüfen Sie Ihre Hypothese aus b), indem Sie die tatsächliche Anzahl der im verwendeten Computer vorhandenen Rechenkerne recherchieren, z. B. durch einen Blick in die vom Betriebssystem bereitgestellten Systeminformationen. Bei Verwendung von schuleigenen Computern kann Ihnen ggf. auch Ihre Lehrkraft eine entsprechende Auskunft geben.

4 Teamarbeit in der Schule

Im Rahmen einer Gruppenarbeit soll ein Referat zum Thema Nebenläufigkeit vorbereitet und mithilfe geeigneter Visualisierungen präsentiert werden. Die Gruppe besteht dabei aus fünf Personen.

- **a** Diskutieren Sie in Kleingruppen, wie sich diese Aufgabe sinnvoll auf die fünf Teammitglieder aufteilen lässt, sodass diese anschließend möglichst parallel arbeiten können.
- **b** Skizzieren Sie einen möglichst optimalen Arbeitsplan, der in fünf Spalten die jeweiligen Tätigkeiten der einzelnen Gruppenmitglieder in chronologischer Reihenfolge aufführt. Markieren Sie ebenfalls in diesem Plan, an welcher Stelle es zu Wartezeiten für ein Gruppenmitglied kommen kann, z. B. weil zunächst auf die Vorarbeit eines anderen Gruppenmitgliedes gewartet werden muss, bevor die nächste Aufgabe begonnen werden kann.
- **c** Ein Mitglied der Fünfergruppe behauptet: „Wenn wir zehn Personen in der Gruppe wären, dann wären wir doppelt so schnell fertig!" Diskutieren Sie diese Aussage; beziehen Sie dabei auch Ihre persönlichen Erfahrungen mit Gruppenarbeiten mit ein.

5 Parallelität beim Waffeln backen

Die SMV möchte in der großen Pause Waffeln verkaufen. Da die Waffeln frisch sein sollen und vor der Pause Unterricht ist, haben die beteiligten Schülerinnen und Schüler lediglich 15 Minuten Zeit, um den Waffelteig anzurühren, die Waffeln zu backen und zu verkaufen. Einige SMV-Mitglieder machen sich daher Sorgen, dass sie dem zu erwartenden Ansturm nicht gewachsen sein könnten, da bestimmt 60 Schülerinnen und Schüler eine Waffel kaufen wollen. Für das Herstellen von 60 Waffeln muss zunächst der Teig angerührt werden. Hierfür besitzt die SMV leider nur eine einzige Rührmaschine, sodass das Anrühren einer Portion Teig für 30 Waffeln stets 5 Minuten dauert. Danach wird der Teig auf die vorhandenen Waffelbäckerinnen und Waffelbäcker verteilt, welche parallel mit dem Backen der Waffeln beginnen. Pro Waffel müssen dann noch zwei Minuten Backzeit eingeplant werden.

- **a** Ermitteln Sie rechnerisch, mit wie vielen Waffeleisen die SMV arbeiten müsste, um innerhalb der 15 Minuten Pause insgesamt 60 Waffeln zu produzieren.
- **b** Erläutern Sie, in welchem Maße sich die Waffelproduktion durch folgende Maßnahmen steigern lässt (jeweils für sich betrachtet):
  - i Anschaffung eines zweiten Rührgerätes bei gleicher Waffeleisenanzahl wie bei a)
  - ii Verwendung eines Fertigteiges aus der Flasche, der nicht angerührt werden muss
  - iii Verdoppelung der Waffeleisenanzahl im Vergleich zu a).

6 Forschungsauftrag: Top 500 Supercomputer

Zweimal im Jahr wird eine „Top 500"-Liste der weltweit leistungsstärksten Supercomputer veröffentlicht.

- **a** Informieren Sie sich im Internet, für welche Zwecke derartige Supercomputer weltweit im Einsatz sind.
- **b** Recherchieren Sie für den aktuell weltweit erstplatzierten sowie für den bestplatzierten deutschen Supercomputer, wie viele Rechenkerne insgesamt in diesen verbaut sind, und vergleichen Sie diese Zahl mit aktuell verfügbaren Computern und Smartphones im Privatbereich.

4.2 Wechselseitiger Ausschluss

Die SMV möchte einen Schulball veranstalten. Hierfür sollen im Vorfeld Getränke, Snacks und Dekorationsartikel eingekauft werden. Für diese Einkäufe steht ein Budget von 500 Euro bereit. Der Kassenwart der SMV schlägt für einen möglichst effizienten Ablauf folgendes Vorgehen vor:

- Der Einkauf wird auf mehrere SMV-Mitglieder aufgeteilt (ein Mitglied kauft Getränke, ein anderes Mitglied die Snacks usw.).
- Damit das Budget nicht überschritten wird, müssen sich alle einkaufenden Mitglieder zunächst beim Kassenwart erkundigen, wie viel Geld noch übrig ist, bevor sie zu ihrem Einkauf aufbrechen.
- Nachdem die Einkäufer zurückkehren, reichen sie ihre Quittung beim Kassenwart ein und erhalten ihre Auslagen erstattet. Das Restbudget verringert sich dadurch entsprechend.

Gesagt getan! Doch es stellt sich heraus, dass die Einkäufer das Budget von 500 Euro bei weitem überschritten haben, obwohl sich alle an das vereinbarte Vorgehen gehalten haben.

a Erläutern Sie, wie es zu der Budgetüberschreitung kommen konnte.

b Stellen Sie einen möglichen Verlauf, der zur Budgetüberschreitung führt, als Sequenzdiagramm dar. Dieses soll neben dem Kassenwart zwei Einkäufer enthalten.

c Diskutieren Sie verschiedene Möglichkeiten, wie das Verfahren angepasst werden könnte, um eine Budgetüberschreitung vollständig auszuschließen.

Signale sorgen bei der Eisenbahn für Sicherheit

Im Alltag kennt man nebenläufige Abläufe z. B. von der Eisenbahn. Im Schienennetz bewegen sich zur gleichen Zeit viele verschiedene Züge weitgehend unabhängig voneinander. Um Unfälle zu vermeiden, muss jedoch sichergestellt werden, dass ein Streckenabschnitt nur von einem Zug gleichzeitig befahren wird. Dafür kommen Signale entlang der Bahnstrecke zum Einsatz, welche die Einfahrt in einen Streckenabschnitt nur dann gestatten, wenn dort noch kein Zug fährt. Nach der Einfahrt eines Zuges schalten die Signale deshalb auf Rot und zwingen so alle eventuell folgenden Züge zum Warten, bis der erste Zug den Streckenabschnitt wieder verlassen hat. Erst dann wird das Signal auf Grün zurückgesetzt, sodass der nächste Zug in den Abschnitt einfahren kann.

Synchronisation von Threads und Prozessen

Auch bei nebenläufigen Programmen kommt es häufig vor, dass etwa bestimmte Ressourcen nur von einem Thread gleichzeitig genutzt werden können. Ein Beispiel hierfür ist die in Smartphones verbaute Hauptkamera. Würden mehrere Threads gleichzeitig auf die Kamera zugreifen, könnte es schnell passieren, dass ein Thread der Kamera den Befehl zum Hereinzoomen gibt, während zeitgleich ein anderer Thread den Befehl zum Herauszoomen gibt. Aber auch beim unkontrollierten nebenläufigen Zugriff auf gemeinsam genutzte Speicherinhalte kann es zu Problemen kommen, wie das folgende Beispiel einer vereinfachten Bankkontoverwaltung zeigt. Die Klasse BANKKONTO fungiert dabei als Schnittstelle zur Datenbank mit den Kontoinformationen,

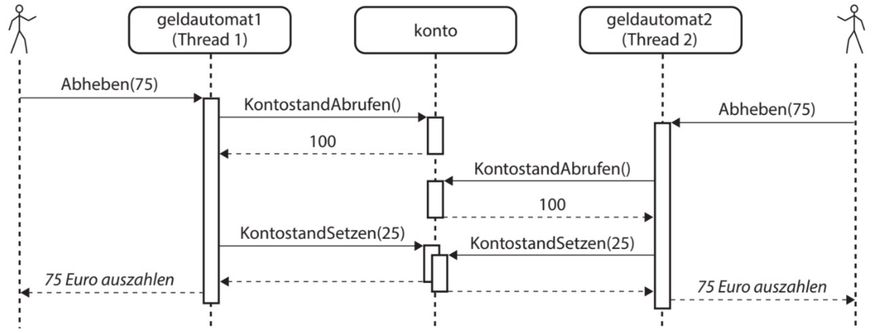
während die Interaktion mit Kunden an einem Bankterminal in der Klasse GELDAUTOMAT realisiert wird. Um eine möglichst flüssige Bedienung zu ermöglichen, wird dabei jeder Methodenaufruf in einem eigenen Thread abgewickelt.

BANKKONTO
…
KontostandAbrufen() KontostandSetzen(betrag)

GELDAUTOMAT
…
Einzahlen(betrag) Abheben(betrag)

```
methode Abheben(betrag)
  kontostand = bank.KontostandAbrufen()
  falls kontoststand >= betrag
    bank.KontostandSetzen(kontostand - betrag)
    // Geld auszahlen
  endefalls
endemethode
```

Betrachtet man das auf einem zentralen Bankserver verwaltete gemeinsame Bankkonto eines Paars mit einem Kontostand von 100 Euro, dessen Inhaber beide gleichzeitig an zwei Geldautomaten 75 Euro abheben möchten, kann es zu folgendem zeitlichen Ablauf kommen:



Um derartige Fehler zu verhindern, muss sichergestellt werden, dass die Methode *Abheben* stets nur von höchstens einem Thread gleichzeitig durchlaufen wird. Allgemein werden Abschnitte im Programmcode, die stets nur von einem Thread gleichzeitig durchlaufen werden dürfen, als **kritische Abschnitte** bezeichnet. Für diese Abschnitte muss dann jeweils ein Verfahren zum **wechselseitigen Ausschluss** der Threads implementiert werden, welches ähnlich wie die Signale bei der Bahn sicherstellt, dass während der Ausführung des kritischen Abschnittes durch einen Thread alle weiteren Threads vorübergehend von der Ausführung dieses Abschnittes ausgeschlossen werden.

Absprachen allein genügen nicht

Bereits in den 60er-Jahren des vergangenen Jahrhunderts wurde nach Lösungen für das Problem der Synchronisation nebenläufiger Abläufe gesucht. Erste Ansätze basierten auf „Absprachen", d. h. gemeinsam genutzten Variablen, die signalisierten, ob ein kritischer Abschnitt aktuell frei oder benutzt ist. Leider ist dieses Verfahren nicht sicher, da zwischen dem Test auf Benutzbarkeit und dem Sperren des Abschnittes für andere Zeit vergeht. So ist es möglich, dass ein Thread feststellt, dass ein kritischer Abschnitt aktuell frei ist, ehe er ihn aber für andere sperren kann, stellt ein anderer Thread ebenfalls fest, dass der Abschnitt frei ist. Daraufhin arbeiten beide Threads verbotenerweise gleichzeitig innerhalb des kritischen Abschnitts. →Dijkstra konnte zeigen, dass eine Lösung des Synchronisationsproblems auf Programmebene unmöglich ist; dazu ist eine Unterstützung durch den Prozessor nötig. Das Betriebssystem muss spezielle Methoden zur Synchronisation bereitstellen, welche selbst →atomar sind. Dies bedeutet, dass diese Methoden stets nur als Ganzes von genau einem Thread gleichzeitig ausgeführt werden können.

→ Edsger Wybe Dijkstra; niederländischer Informatiker, 1930–2002

→ von altgr. atomos: unteilbar

→gr. sema: Zeichen, gr. pherein: tragen; in etwa „Zeichenträger". Auch klassische Bahnsignale werden historisch als Semaphore bezeichnet und im Spanischen bezeichnet „semáforo" eine Verkehrsampel.

Das Prinzip dieser Nutzungsmethoden (sog. →Semaphorprinzip) funktioniert wie digitale Bahnsignale.

Monitore vereinfachen den wechselseitigen Ausschluss

Häufig beschränken sich **kritische Abschnitte** jedoch nicht nur auf einen eng eingrenzbaren Teil des Programmcodes wie etwa eine Methode. Bei der bereits als Beispiel angeführten Klasse GELDAUTOMAT bilden beispielsweise die Methoden *Abheben* und *Einzahlen* pro Konto einen gemeinsamen kritischen Abschnitt, da nicht nur zwei parallel ausgeführte Abhebungen von einem Konto zu Fehlern führen können, sondern ebenso die parallele Ausführung einer Abhebung und einer Einzahlung. Eine zentrale Herausforderung beim Entwickeln nebenläufiger Programme ist es daher, alle kritischen Abschnitte als solche zu identifizieren und entsprechend abzusichern.
Viele Programmiersprachen bieten hierfür sogenannte **Monitore** an. Dabei werden die zu sichernden Codeabschnitte lediglich mit einer speziellen Markierung versehen, um diese als kritische Abschnitte zu kennzeichnen. Sämtliche so markierten Codeabschnitte bilden damit pro Objekt einen gemeinsamen kritischen Abschnitt. Während der Programmausführung wird die Synchronisation der Threads dann automatisch im Hintergrund durchgeführt und sichergestellt, dass stets höchstens ein Thread irgendeinen der entsprechend markierten Codeabschnitte durchläuft.

Python

```
import threading
lock = threading.Lock()
...
with lock:
    # kritischer Abschnitt
```

Swift

```
let monitor = NSCondition()
...
monitor.lock()
    // kritischer Abschnitt
monitor.unlock()
```

Java

```
...
synchronized {
    // kritischer Abschnitt
}
```

Markierung eines kritischen Abschnitts

Häufig werden dabei mehrere Methoden einer Klasse über den gleichen Monitor abgesichert, sodass für ein Objekt der Klasse alle diese Methoden einen gemeinsamen kritischen Abschnitt bilden.

Abschnitte im Programmcode, die stets nur von maximal einem Thread gleichzeitig durchlaufen werden dürfen, werden **kritische Abschnitte** genannt. Für diese Abschnitte muss der **wechselseitige Ausschluss** z. B. durch **Monitore** sichergestellt werden.

Aufgaben

1 Die Bankkontoverwaltung

Betrachten Sie das Beispiel der Bankkontoverwaltung auf S. 122–123.

a Wie im Lehrtext erwähnt, kann es auch zu einem inkonsistenten Datenbestand kommen, falls gleichzeitig die Methoden *Abheben* und *Einzahlen* für ein Konto aufgerufen werden. Zeichnen Sie ein Sequenzdiagramm für einen solchen fehlerhaften Ablauf und beschreiben Sie anhand des Diagramms, weshalb es dabei zu einem inkorrekten Kontostand kommt.

b Diskutieren Sie, inwieweit ein korrekter nebenläufiger Aufruf der Methoden *Abheben* und *Einzahlen* auch gewährleistet werden könnte, indem unmittelbar vor dem Aufruf von *KontostandSetzen* der aktuelle Kontostand noch einmal mittels *KontostandAbrufen* geprüft wird.

Mithilfe der Sequenzdiagramme lässt sich sehr zielgerichtet herausfinden, wo Nebenläufigkeiten, also die mögliche parallele Nutzung eines Objekts durch zwei parallele Prozesse, auftreten können.

2 Monitore zur Threadsynchronisation einsetzen

In dem zu dieser Aufgabe bereitgestellten Programm wird eine Wetterstation simuliert. Das Programm arbeitet dabei mit mehreren Threads, wobei jede erfasste Messgröße (Temperatur, Luftdruck usw.) von einem eigenen Thread erfasst und als Text ausgegeben wird.

a Identifizieren Sie alle kritischen Abschnitte im vorgegebenen Programmcode.

b Ergänzen Sie den Programmcode sinnvoll, damit fehlerhafte Ausgaben zukünftig vermieden werden. Verwenden Sie zur Synchronisation der Threads das Monitorkonzept.

3 Nebenläufigkeit bei Datenbanken

Datenbankserver erlauben in der Regel mehreren Nutzern den nebenläufigen lesenden und schreibenden Zugriff auf ihre Datenbestände.

a Erläutern Sie an einem selbst gewählten Beispiel, wie es durch den nebenläufigen Zugriff zweier Nutzer zu inkonsistenten Daten in einer Datenbanktabelle kommen kann.

b Erklären Sie, wie die in a) genannten Inkonsistenzen durch das Prinzip des wechselseitigen Ausschlusses verhindert werden könnten.

c Wechselseitiger Ausschluss kann auf Datenbankebene, Tabellenebene oder auch Datensatzebene realisiert werden, wobei sichergestellt wird, dass entweder die komplette Datenbank, einzelne Tabellen oder einzelne Datensätze einer Tabelle nur von einem Nutzer gleichzeitig genutzt werden können. Beurteilen Sie, welche Art von wechselseitigem Ausschluss für die erfolgreiche Ausführung der folgenden Operationen mindestens gewährleistet sein muss:

- i Anlegen einer neuen Tabelle
- ii Ändern des Wertes eines Attributs in einem Datensatz, das nicht Teil eines Schlüssels ist
- iii Ändern des Wertes eines Primärschlüsselattributs in einem Datensatz
- iv Einfügen eines neuen Datensatzes in einer Tabelle

d Diskutieren Sie, inwiefern es beim parallelen Zugriff auf eine Datenbanktabelle zu Inkonsistenzen kommen kann, wenn Datensätze ausschließlich gelesen und hinzugefügt, aber nie verändert oder gelöscht werden. Gehen Sie dabei davon aus, dass das Einfügen eines Datensatzes atomar erfolgt.

e Für Schnelle: Viele Datenbanksysteme verzichten auf das Erzwingen eines vollständigen wechselseitigen Ausschlusses und arbeiten stattdessen mit voneinander isolierten „Transaktionen". Recherchieren Sie die Funktionsweise dieser Transaktionsisolation und diskutieren Sie Vor- und Nachteile dieses Vorgehens im Vergleich zu einem strikten wechselseitigen Ausschluss, bei dem stets nur ein Nutzer zeitgleich Zugriff auf eine Datenbank oder Teile davon erlangen kann.

4 Ist ein wechselseitiger Ausschluss immer notwendig?

Ein nebenläufiges Programm besteht aus zehn Threads und einer von allen Threads gemeinsam genutzten Variable. Die Lese- und Schreibzugriffe auf die Variable erfolgen für sich genommen atomar. Diskutieren und beurteilen Sie in Kleingruppen, in welchen der folgenden Fälle darüber hinaus ein wechselseitiger Ausschluss zur Vermeidung inkonsistenter Daten benötigt wird:

i Alle Threads greifen stets nur lesend auf die gemeinsame Variable zu.

ii Nur genau ein Thread greift lesend und schreibend auf die Variable zu, alle übrigen Threads lesen den Variableninhalt nur.

iii Neun Threads greifen nur schreibend (niemals lesend) auf die Variable zu; der zehnte Thread greift lesend und schreibend auf die Variable zu.

iv Fünf Threads greifen nur schreibend (niemals lesend) auf die Variable zu; die übrigen fünf Threads greifen hingegen nur lesend auf die Variable zu.

v Alle Threads greifen sowohl lesend als auch schreibend auf die Variable zu.

5 Synchronisation mittels einer einfachen Variable

Lena hat versucht, den Zugriff auf den kritischen Abschnitt mit Hilfe einer Booleschen Variablen zu koordinieren:

```
attribut istFrei: WAHRHEITSWERT = WAHR
…
methode KontostandVerzinsen(zinssatz)
        falls istFrei dann
            istFrei = FALSCH
            kontostand = kontostand * (1 + zinssatz)
            istFrei = WAHR
        endefalls
endemethode
```

Trotzdem liefert ihr Programm immer wieder falsche Ergebnisse.

Erläutern Sie unter Angabe eines Sequenzdiagramms, warum Lenas Konzept nicht funktioniert.

6 Der Paketdienst (aus Abitur 2017)

Ein auf Paketversand spezialisiertes Unternehmen hat flächendeckend an verschiedenen Orten Paketstationen aufgestellt. […] Der Paketdienst bietet die Möglichkeit, dass auch ein Privatkunde fertig adressierte und frankierte Pakete zulässiger Größe in der Paketstation ablegen kann, die dann von einem Mitarbeiter abgeholt werden. Hierzu muss der Kunde im Voraus ein Ablagefach der Paketstation online reservieren. Auf dem hierfür verwendeten Server des Paketdienstes wird dies durch folgenden Algorithmus abgewickelt:

```
falls in der Paketstation noch ein Ablagefach frei ist
dann
    nimm die Reservierungen an
    verringere die Zahl der freien Ablagefächer um eins
    weise ein freies Ablagefach zu
endefalls
```

Die Anzahl der reservierbaren Ablagefächer in einer konkreten Paketstation beträgt 15. Erläutern Sie, wie es geschehen kann, dass der Server mehr als 15 Reservierungen gleichzeitig gespeichert hat, und stellen Sie eine Maßnahme dar, dies zu verhindern.

7 Der Ticketverkauf

Eine Eventagentur verkauft Tickets für eine Konzertveranstaltung über einen Online-Shop, der jeden Bestellvorgang in einem eigenständigen Thread abwickelt. Im Konzertsaal gilt freie Platzwahl, aber es muss sichergestellt werden, dass keinesfalls mehr Tickets verkauft werden, als Plätze vorhanden sind.

a Gehen Sie zunächst davon aus, dass keinerlei Maßnahmen zur Synchronisation der Verkaufsvorgänge ergriffen wurden. Spielen Sie in Kleingruppen einen Ablauf als Rollenspiel nach, bei dem es zu einer Überbuchung kommt.

b Geben Sie basierend auf a) an, welche Teile des Online-Shopsystems einen kritischen Abschnitt bilden.

c Bisher konnten Tickets im Online-Shop nur direkt in einem Schritt gekauft werden. Nun soll stattdessen ein Warenkorbsystem eingeführt werden. Nachdem ein Ticket in den Warenkorb gelegt wurde, ist es maximal 15 min für den jeweiligen Kunden reserviert. Wird der Kauf in dieser Zeit nicht abgeschlossen, endet die Reservierung automatisch und das Ticket wird aus dem Warenkorb wieder entfernt. Im Shop wird dies intern durch die Methoden *TicketReservieren*, *TicketkaufAbschliessen* und *ReservierungStornieren* abgebildet. Beurteilen Sie begründet, welche dieser Methoden einen kritischen Abschnitt enthalten.

8 Debugging

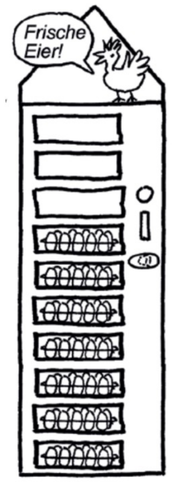
Fehlfunktionen in Softwareprodukten, die auf eine unzureichende oder inkorrekte Synchronisation nebenläufiger Abläufe zurückzuführen sind, gelten als besonders schwer auffindbar.

a Erklären Sie allgemein, wie es dazu kommen kann, dass ein nebenläufiges Programm im produktiven Einsatz fehlerhaft funktioniert, obwohl es während einer vorangegangenen Testphase keinerlei Fehlfunktion zeigte.

b Erstellen Sie gemeinsam in Kleingruppen eine Checkliste mit konkreten Handlungsempfehlungen für die Arbeit mit nebenläufigem Programmcode, damit Fehler im Code bestmöglich vermieden werden können. Beziehen Sie in Ihre Überlegungen neben der unmittelbaren Arbeit mit dem Programmcode auch die Aspekte Modellierung und Visualisierung mit ein.

c Um „auf Nummer sicher zu gehen", dass es nicht zu Fehlern durch unzureichende Threadsynchronisation kommt, schlägt eine Kollegin bei einem Softwareprojekt vor, einfach alle nebenläufigen Methoden sicherheitshalber durch einen Monitor zu schützen. Erläutern Sie, weshalb dies in der Regel keine gute Idee ist.

4.3 Eine gemeinsame Ressource: Effiziente Synchronisierung



Ein kleiner Eierautomat auf dem Lande fasst zehn Eierkartons. Um ihre Buchhaltung zu vereinfachen, befüllt die Landwirtin den Automaten nur, wenn er ganz leer ist. Da sie aber auch oft anderweitig beschäftigt ist, kann sie das Gerät nicht ständig kontrollieren. Der Kunde Leo braucht sehr häufig einen Karton Eier zum Backen; findet er den Automaten leer vor, kommt er oft mehrmals täglich wieder, bis der Automat endlich wieder befüllt wurde.

a Beschreiben Sie, wie die Abläufe beim Eierautomaten unter den gegebenen Voraussetzungen optimiert werden könnten.

b Im vorliegenden Projekt wurde die beschriebene Situation implementiert. Erstellen Sie ein Objekt der Klasse LANDLEBEN und bestimmen Sie auf Basis einer Momentaufnahme ein ungefähres Verhältnis von Befüllungen der Bäuerin zu Eierholbesuchen von Leo.

c In der Klasse BESSERER_EIERAUTOMAT ist ein verbessertes Verfahren in der Methode *Befuellen()* bereits umgesetzt. Analysieren Sie die Implementierung und ergänzen Sie auch in *EierHolen()* eine entsprechende Strategie an den markierten Stellen. Passen Sie den Objekttyp in LANDLEBEN an und testen und bewerten Sie dann Ihre Lösung.

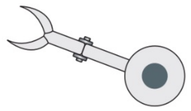
d Für Schnelle: Im Winter kann die Bäuerin den Automaten häufiger kontrollieren, während Leo dann deutlich weniger backt. Setzen Sie dies durch Modifikation der Zeiten um, testen Sie erst mit einem normalen, dann mit einem besseren Eierautomaten und überprüfen Sie, ob Ihre Lösung auch in diesem umgekehrten Szenario funktioniert.

Schwierigkeiten bei der gemeinsamen Nutzung einer Ressource

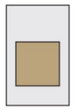
In den bisher betrachteten Fällen wie z. B. beim gemeinsamen Konto genügte es, den Zugriff nebenläufiger Threads auf die gemeinsam genutzte Ressource durch den Einsatz von Monitoren zu koordinieren: Hatte ein Thread den Zugriff auf den kritischen Abschnitt erhalten, konnte er die Ressource immer unmittelbar nutzen, seine Aufgabe erledigen und dann den kritischen Abschnitt wieder verlassen; der Zustand der gemeinsam genutzten Ressource war dabei unerheblich. Dadurch traten für die Threads weder nennenswerte Wartezeiten auf noch waren logistische Überlegungen zu ihrer Zugriffsreihenfolge erforderlich. Im Folgenden werden nun Problemklassen illustriert, bei denen diese Aspekte eine Rolle spielen.

Erzeuger-Verbraucher-Probleme: Ressourcenzustand verursacht Wartezeiten

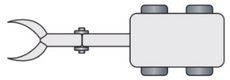
Ein Roboter füllt Kisten und legt sie dann auf einem Zwischenspeicherplatz ab. Ein zweiter Roboter holt die Kisten von diesem Zwischenspeicherplatz und ordnet sie in ein Regallager ein. Während der erste Roboter mit gleichbleibender Geschwindigkeit arbeitet, benötigt der zweite Roboter mal mehr, mal weniger Zeit, je nach Entfernung der Lagerplätze für die Kisten. Auf den Zwischenspeicherplatz darf immer nur ein Roboter zugreifen, damit sie sich nicht gegenseitig beschädigen. Die beschriebene Situation wird allgemein als **Erzeuger-Verbraucher-Problem** bezeichnet und ist eine beispielhafte Beschreibung von Problemen, bei denen sich erzeugende und verbrauchende Threads einen gemeinsamen Zwischenspeicher teilen.



Roboter 1 (Erzeuger)

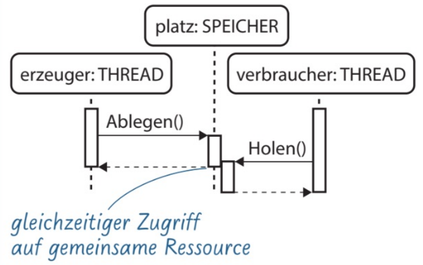


Zwischenspeicher mit Ware

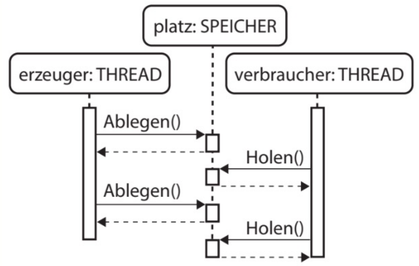


Roboter 2 (Verbraucher)

Das zugehörige Sequenzdiagramm spiegelt die Nebenläufigkeit beim Zugriff auf den Zwischenspeicher wider:



Werden die Methoden *Ablegen* und *Holen* als Monitor erklärt, scheint das Problem auf den ersten Blick behoben, *Ablegen* und *Holen* wechseln sich einfach ab:



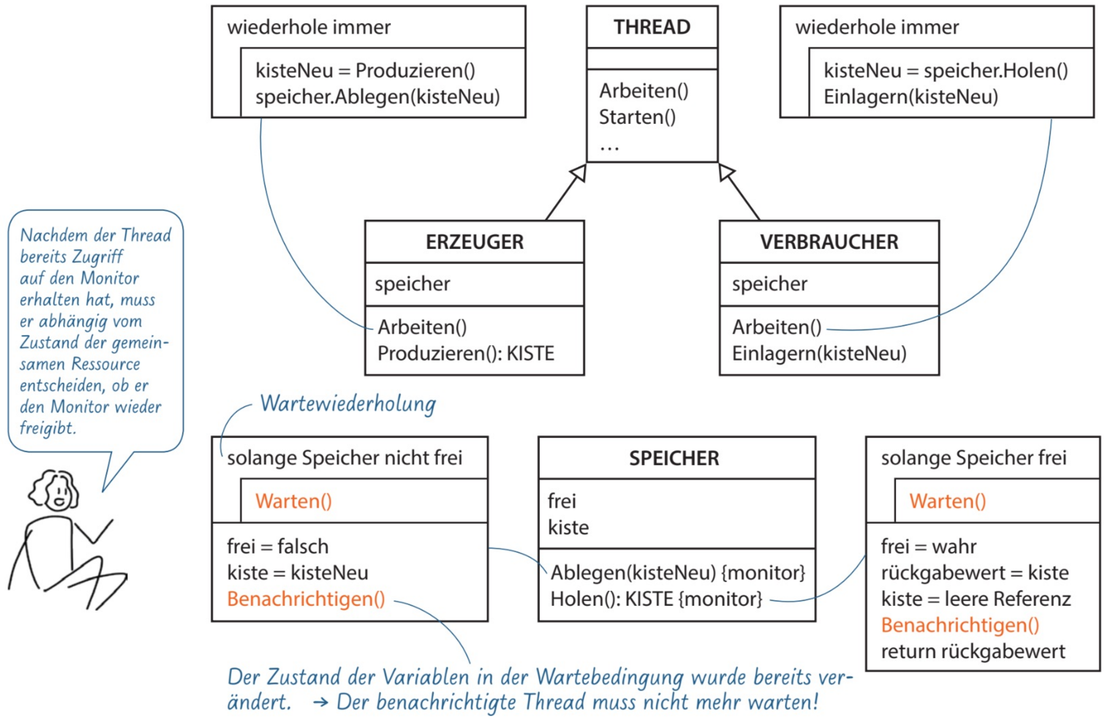
Da aber beide Roboter unterschiedlich schnell arbeiten, kann es zum Beispiel passieren, dass der Erzeuger Kiste 5 schon verpackt hat, noch ehe der Verbraucher Kiste 4 abholen konnte. Der Erzeuger kann dann Kiste 5 nicht ablegen, da der Platz ja noch besetzt ist. Er kann aber auch nicht einfach mit dem Arm über Kiste 4 warten, da sonst der Verbraucher diese Kiste nicht abholen kann.

Lösung durch Absprachen

Der Erzeuger kann nun einfach den Arm zurückziehen, etwas warten und später immer wieder aktiv versuchen, den Monitor zu betreten (**aktives Warten**/busy waiting). Das Problem ist damit sicher gelöst, aber mit unnötig viel Aufwand: Sind die Wartezeitintervalle kurz, muss es der Erzeuger unter Umständen öfter versuchen, bis er die Kiste ablegen kann, und er hindert bei diesen Versuchen vielleicht sogar den Verbraucher am Abholen. Sind die Wartezeitintervalle lang, vergeudet der Erzeuger Arbeitszeit. Ähnliches gilt für den Verbraucher. Viel effizienter ist es, wenn der Verbraucher dem Erzeuger mitteilen kann, dass er jetzt die Kiste abgeholt hat und damit der Ablageplatz wieder frei ist. Eine Erweiterung des Monitorkonzeptes um zwei Methoden ermöglicht diese Kommunikation. Beide Methoden können nur innerhalb von Monitoren verwendet werden.

Warten()	Der Thread gibt den Monitor frei und wird in eine Warteschlange für diesen Monitor eingereiht.
Benachrichtigen()	Das Betriebssystem wählt einen beliebigen der auf den aktuellen Monitor wartenden Threads aus. Sobald der aktive Thread den Monitor endgültig verlassen hat, bekommt der benachrichtigte Thread Zugriff auf den Monitor.

Damit kann der Erzeuger nun, wenn der Platz belegt ist, genau so lange warten, bis der Verbraucher sagt, dass der Platz frei ist, und umgekehrt (**passives Warten**). Das Vorgehen ist hinsichtlich einer Minimierung von „Betreteversuchen“ und Wartezeiten maximal effizient. Die Klassen ERZEUGER, VERBRAUCHER und SPEICHER haben folgendes typisches Aussehen:



Java	Python	Swift
	`import threading`	`let monitor = NSCondition()`
`synchronized … {`	`lock = threading.Condition()`	`...`
`  try {wait();}`	`...`	`monitor.lock`
`  catch (…){ }`	`with lock:`	`monitor.wait()`
`  …`	`  lock.wait()`	`...`
`  notify();`	`  ...`	`monitor.signal()`
`}`	`  lock.notify()`	`monitor.unlock`

Erzeuger-Verbraucher-Probleme sind Beispiele für Synchronisationsprobleme, bei denen es vom Zustand der gemeinsam genutzten Ressource abhängt, ob ein Thread seine Aufgabe erledigen kann. Innerhalb eines Monitors kann der Zugriff durch Absprachen koordiniert werden (**passives Warten**): Die Methoden ***Warten*** und ***Benachrichtigen*** (bzw. *AlleBenachrichtigen*) realisieren die Kommunikation zwischen den Threads. So wird eine permanente Neuüberprüfung des Ressourcenzustands (**aktives Warten**) vermieden.

Aufgaben

1 Richtig oder falsch?
Entscheiden Sie zu jeder Aussage, ob sie richtig oder falsch ist. Begründen und berichtigen Sie bei falschen Aussagen.
- **a** Sobald ein Thread beim Erzeuger-Verbraucher-Problem Zugriff auf den Monitor erhalten hat, kann er seine Aufgabe erledigen.
- **b** Passives Warten bedeutet, regelmäßig zu versuchen, den Monitor zu betreten.
- **c** Beim Aufruf von *Benachrichtigen()* erhält der erste Thread in der Warteschlange sofort Zugriff auf den Monitor.

2 Warten im Alltag
Illustrieren Sie die Strategien „Aktives Warten" und „Passives Warten" erst im Sachkontext eines Pakets, das Sie sehnlichst erwarten, und erklären Sie die Konzepte dann allgemein.

3 Firmenkonto
Eine kleine Firma hat für ihr Firmenkonto keinen Dispokredit: Es dürfen also nur Abhebungen vorgenommen werden, die den Kontostand nicht unter Null sinken lassen.
In einer vereinfachten Simulation könnten Einzahlungen und Abhebungen mit Hilfe von Threads umgesetzt und das Konto als gemeinsam genutzte Ressource mit synchronisierten Zugriffsmethoden implementiert werden.
Erarbeiten Sie in Vierergruppen ein kurzes Rollenspiel, das die Abläufe bei zufälligen Ein- und Auszahlungsvorgängen darstellt. Mögliche Rollenverteilung: Einzahler, Auszahler, Konto, Moderator der Abläufe

4 Boote am Chiemsee
Ein hochfrequentierter Steg am Chiemsee bietet 10 Liegeplätze für Boote. Diese müssen vor dem Anlegen per App reserviert und beim Ablegen wieder freigegeben werden. Dazu stehen Methoden in der Klasse STEGVERWALTUNG zur Verfügung, die die Boote intern in einem Feld verwaltet.

- **a** Ermitteln Sie anhand des Quellcodes, weshalb es zu Inkonsistenzen bei der Liegeplatzvergabe kommen kann. Stellen Sie das Problem in einem Sequenzdiagramm dar. Geben Sie dann eine Verbesserung des Quellcodes an, die das Problem behebt.
- **b** Häufig sind alle Plätze belegt und die Reservierungsmethode muss auf einen freien Platz warten. Erklären Sie anhand des Quellcodes, mit welcher Strategie dies bisher umgesetzt wurde und warum die vorliegende Umsetzung nach Ihrer Verbesserung aus a) nicht mehr funktioniert.
- **c** Setzen Sie eine andere Wartestrategie um, die weniger ressourcenintensiv ist.

5 Erzeuger-Verbraucher-Problem

Im vorliegenden Projekt ist das Erzeuger-Verbraucher-Problem implementiert. In den Klassen ERZEUGER und VERBRAUCHER wird das Produzieren bzw. Einlagern in Millisekundenschritten simuliert.

a Der Zugriff von Erzeuger und Verbraucher auf die gemeinsame Ressource wurde im Projekt bereits synchronisiert. Benennen und erklären Sie das Konzept, das dabei umgesetzt wurde.

b Betrachten Sie die grafische Darstellung im Vorlagenprojekt.

- i Erklären Sie anhand der grafischen Darstellung, welche Strategie zum Warten auf einen freien Ablageplatz im Projekt verwendet wird.
- ii Verifizieren Sie Ihre Vermutung anhand des Quellcodes. Erklären Sie, warum die Strategie ungünstig ist. Begründen Sie auch, warum für die Umsetzung ein Rückgabewert der Methode *Ablegen()* notwendig ist.
- iii Ergänzen Sie eine Variable zum Mitzählen der Vergleiche in den Methoden von SPEICHER und geben Sie deren Wert aus.

c Verbessern Sie den Quellcode in einer Kopie des Projekts: Sorgen Sie dafür, dass die Threads passiv warten. Vergleichen Sie dann – für eine etwa gleich große Zeitspanne – die vorherige Anzahl von Vergleichen mit der jetzigen.

Arbeiten Sie im Folgenden in der Projektvorlage ohne grafische Darstellung.

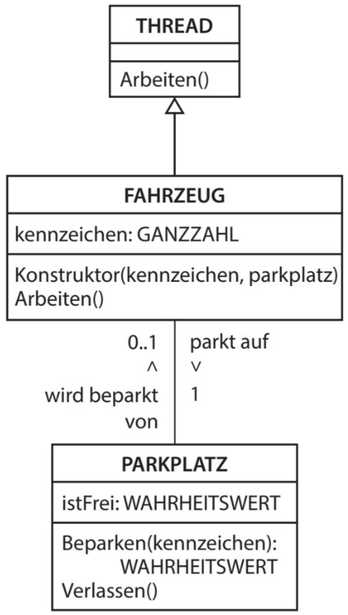
d Zählen Sie mit einer Variablen, wie häufig *Warten()* vom Erzeuger- bzw. Verbraucherthread aufgerufen wird. Erweitern Sie dann die Anzahl der verfügbaren Ablegeplätze in SPEICHER mit Hilfe eines Felds auf 10 und vergleichen Sie die Variablenwerte für eine etwa gleich große Zeitspanne. Erklären Sie Ihre Beobachtung.

e Erweitern Sie auf fünf Erzeuger und fünf Verbraucher. Vergessen Sie nicht, die Benachrichtigung in den Methoden von SPEICHER anzupassen. Vergleichen Sie die Werte Ihrer Variablen zum Zählen der Vergleiche bzw. *Warten*-Aufrufe und ordnen Sie Ihre Beobachtung ein.

f Für Schnelle: Implementieren Sie in einer Kopie des Projekts eine Umsetzung mit Semaphoren. Die Zugriffsversuche beim aktiven Warten sollen mit einem Zeitabstand von 100 ms erfolgen.

6 Parkverbot

In der Innenstadt gibt es eine beliebte Stelle im Parkverbot, die Platz für ein einzelnes Fahrzeug bietet. Für schnelle Erledigungen werden Fahrzeuge dort häufig bis zu fünf Minuten abgestellt. Ist der Platz gerade besetzt, hat es sich etabliert, einmal um den Block zu fahren und es dann erneut zu probieren. Im Folgenden soll der beschriebene Parkverkehr mit Threads simuliert werden.



Beparken(kennzeichen): WAHRHEITSWERT

wahr / istFrei	falsch
istfrei = FALSCH AusgabeZeile(kennzeichen, " erfolgreich eingeparkt.") return WAHR	
AusgabeZeile(kennzeichen, " fährt um den Block.") return FALSCH	

Verlassen()

```
AusgabeZeile(kennzeichen, " ausgeparkt")
frei = WAHR
```

Um die Ausführung eines Threads für eine bestimmte Zeitspanne zu unterbrechen (Simulation von Parken/um den Block fahren), kann die Methode Schlafen(zeitspanne) genutzt werden.

a Erstellen Sie eine Programmsimulation für das im Text beschriebene Szenario auf Basis des gegebenen Klassendiagramms und der Struktogramme. Erstellen Sie auch eine Rahmenklasse, die zehn Fahrzeugthreads erstellt und startet. Der Einfachheit halber sollen diese zehn Fahrzeuge immer wieder ein- und ausparken.

b Vergleichen Sie Ihr Programm mit der zur Verfügung gestellten Lösung und korrigieren Sie gegebenenfalls. Führen Sie dann die Lösungssimulation aus und begründen Sie anhand der Ausgaben, dass die Lösung noch einen Fehler enthält. Erklären Sie, warum der Fehler z. B. mit einem Monitor vermieden werden kann, und verbessern Sie den Quellcode entsprechend. Testen Sie erneut.

c Diskutieren Sie zu zweit, ob die Situation einer Ihnen bekannten Problemklasse zugeordnet werden kann.

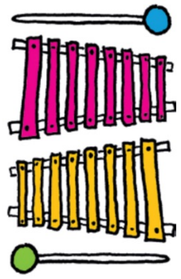
d Mehrmals um den Block zu fahren ist aufwändig und nicht ökologisch. Verbessern Sie das Programm, indem Sie die Strategie „Passives Warten" für die Threads umsetzen. Diskutieren Sie zu zweit, ob sich diese auch auf den Sachkontext der Parkplatzsuche übertragen lässt.

e Anton hat sowohl in *Beparken* als auch in *Verlassen* den Aufruf der Methode *Benachrichtigen()* vergessen. Treffen Sie eine Voraussage, welche Ausgabe bei seinem Programm zu erwarten ist, und überprüfen Sie Ihre Lösung dann.

7 Forschungsauftrag: spurious wakeups

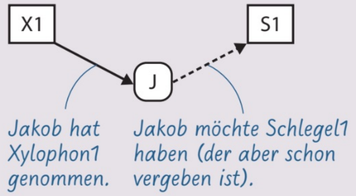
Der Pseudocode im Lehrtext zum passiven Warten verwendet eine Wartewiederholung. Diskutieren Sie zu zweit, ob eine bedingte Anweisung nicht ausreichend wäre. Recherchieren Sie dann zum Begriff „spurious wakeups" und besprechen Sie sich erneut.

4.4 Mehrere gemeinsame Ressourcen: Deadlocks verhindern



Die Kinder Jakob, Matilda und Ina wollen musizieren. Jedes Kind möchte eine Weile mit einem Xylophon und einem Schlegel spielen. Insgesamt sind zwei verschiedenfarbige Xylophone und zwei verschiedenfarbige Schlegel verfügbar. Alle versuchen gleichzeitig, sich die benötigten Gegenstände zu sichern. Hat ein Kind einen Gegenstand mit seiner Wunschfarbe ins Auge gefasst, gibt es sich auch nicht mit einem andersfarbigen gleichartigen Gegenstand zufrieden.

a Beschreiben Sie auf Basis des angegebenen Graphenteils eine Aufteilung der Gegenstände, in der es zu einer festgefahrenen Situation durch eine gegenseitige Blockade kommen kann. Übertragen Sie den Graphen in Ihr Heft und vervollständigen Sie ihn dann entsprechend Ihrer Beschreibung.



b Erklären Sie, woran man die gegenseitige Blockade in Ihrem Graphen erkennen kann.

c Formulieren Sie eine einfache Regel, bei deren Einhaltung ein solches Problem nicht auftreten kann.

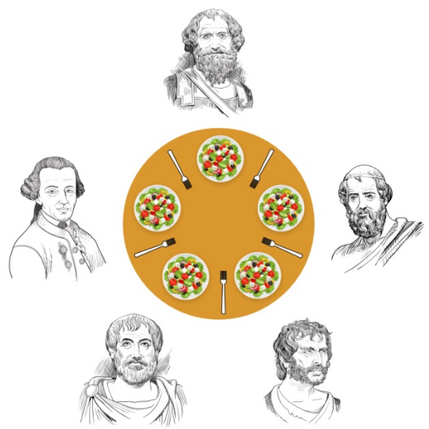
Schwierigkeiten bei der gemeinsamen Nutzung mehrerer Ressourcen

Christian und Sarah wollen am Samstagvormittag zum Mountainbiken. Zu dieser Zeit sind die Plätze im Zubringerbus aber schon fast ausgebucht: Es steht nur noch ein Sitzplatz und ein Fahrradplatz zur Verfügung. Christian legt in der App des lokalen Busunternehmens den Fahrradplatz in seinen Warenkorb und dann den Sitzplatz. Sarah geht genau andersherum vor. Als beide gleichzeitig auf „kostenpflichtig bestellen" tippen, werden auf dem zentralen Buchungsserver zwei Buchungsthreads gestartet: Christians Thread reserviert zunächst sein erstes Warenkorbelement, den Fahrradplatz; Sarahs Thread reserviert den Sitzplatz. Damit sind alle Plätze belegt; die beiden Threads können aber ihre Arbeit nicht beenden, da sie nun darauf warten, den jeweils anderen Platz im Warenkorb buchen zu können. Da sie die Reservierung des ersten Platzes auch nicht wieder aufgeben, blockieren sich die Threads gegenseitig. Solche Probleme, die nur entstehen können, wenn parallele Prozesse mehrere Ressourcen/Betriebsmittel gemeinsam nutzen, werden anschaulich mit dem **Problem der speisenden Philosophen** beschrieben.

Die speisenden Philosophen

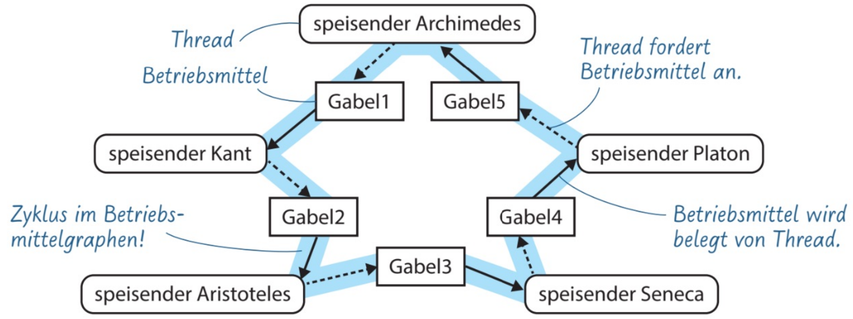
Fünf Philosophen sitzen um einen Tisch, der wie in der Abbildung gedeckt ist. Für das Essen gilt:

- Ein Philosoph benötigt zwei Gabeln, um das servierte Gericht zu verspeisen.
- Ein hungriger Philosoph greift zuerst zur linken Gabel, dann zur rechten, isst und legt einmal aufgenommene Gabeln erst nach beendeter Mahlzeit wieder ab.
- Liegt eine benötigte Gabel nicht an ihrem Platz, wartet ein Philosoph stoisch, bis sie wieder verfügbar ist.



So lange nicht alle Philosophen gleichzeitig zu essen beginnen wollen, führt das Verfahren nur zu gelegentlichen Wartezeiten. Greifen aber alle Philosophen im selben Augenblick zum Besteck, nimmt jeder Philosoph zunächst die links liegende Gabel auf. Dadurch werden allen die rechts liegenden Gabeln durch ihren Nachbarn weggenommen. Da keiner die aufgenommene Gabel wieder ablegt, solange er nicht gegessen hat, blockieren sich die Philosophen gegenseitig und werden verhungern.

Ein **Betriebsmittelzuteilungsgraph** visualisiert, welche Threads (die speisenden Philosophen) welche Ressourcen (die Gabeln) belegen bzw. angefordert haben, und entlarvt die Konstellation als eine **zyklische Wartesituation**, in der jeder beteiligte Thread auf die Freigabe von Betriebsmitteln wartet, die bereits von einem anderen Thread exklusiv belegt werden: Eine **Verklemmung** (**Deadlock**) ist entstanden.



Voraussetzungen für Verklemmungen

Das Beispiel erfüllt alle vier Voraussetzungen, die für eine Verklemmungssituation gegeben sein müssen. Sie werden als →**Coffman-Bedingungen** bezeichnet:

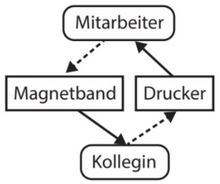
1. **Wechselseitiger Ausschluss (mutual exclusion):** Jedes Betriebsmittel wird entweder von genau einem Thread belegt oder es ist verfügbar.
2. **Halten und Warten (hold and wait):** Ein Thread, der bereits Betriebsmittel belegt, kann weitere Betriebsmittel anfordern.
3. **Ununterbrechbarkeit (non-preemption):** Die von einem Thread belegten Betriebsmittel können nicht von außen entzogen werden; der Thread selbst muss sie freigeben.
4. **Zyklisches Warten (cyclic waiting):** Es gibt eine zyklische Kette von Threads, von denen jeder ein Betriebsmittel angefordert hat, das vom nächsten Thread der Kette belegt ist.

→formuliert 1971 von Coffman, Elphick und Shoshani

Voraussetzung 4 kann nur eintreten, wenn die Bedingungen 1–3 erfüllt sind.

Verklemmungen von vornherein verhindern

Bevor aus Coffmans Analyse der Verklemmungsvoraussetzungen auf breiter Front Konsequenzen gezogen wurden, waren Verklemmungen keine Seltenheit: In einem typischen Firmenbüro der Siebzigerjahre gab es häufig Terminals, von denen aus der Zugriff auf den gemeinsam genutzten →Magnetband-Speicher eines Zentralrechners möglich war. Es konnte dann etwa passieren, dass ein Mitarbeiter von einem Terminal aus eine Datei vom Magnetband drucken wollte; das Druckprogramm reservierte den Drucker, um anschließend die gewünschte Datei zu öffnen. Eine Kollegin hatte das Magnetband allerdings schon exklusiv belegt, weil sie an einer anderen darauf gespeicherten Datei gearbeitet hatte, die sie nun im Texteditor ausdrucken wollte.



→Magnetband: historischer Datenträger, auf den nur exklusiv zugegriffen werden kann

Mittlerweile existieren Verklemmungen wie im Beispiel nicht mehr. Die Exklusivität des Druckerzugriffs wird heutzutage durch Druckerwarteschlangen aufgehoben, welche jedem Thread „seinen eigenen" virtuellen Drucker bieten. Die Strategie verfolgt die Grundidee, dafür zu sorgen, dass einzelne Coffman-Bedingungen (und damit auch alle vier gleichzeitig) nie erfüllt sind; in diesem Beispiel wird Coffman-Bedingung 1 außer Kraft gesetzt. Auch andere Verklemmungen treten so gut wie gar nicht mehr auf; auf Basis der Coffman-Bedingungen hat man allgemeine Strategien entwickelt, um potenzielle Verklemmungssituationen zu verhindern. Je nach Programmsituation muss nur noch analysiert werden, welche der Bedingungen am besten außer Kraft gesetzt werden kann:

- „Halten und Warten" kann vermieden werden, indem jeder Thread alle möglicherweise benötigten Betriebsmittel gleich zu Beginn anfordert. Er bekommt sie dann „auf einmal" zugeteilt, wenn alle gleichzeitig frei sind.
  Das schränkt die gewünschte Parallelität aber deutlich ein.
  Eine gängige Variante ist, alle für einen Teilschritt benötigen Ressourcen auf einmal anzufordern und sie dann nacheinander möglichst schnell wieder freizugeben. Komplexere Methoden erlauben zwar prinzipiell höhere Parallelität, werden aber in der Regel wegen des höheren Verwaltungsaufwands nicht eingesetzt.
  Alternativ kann die Anforderung einer weiteren Ressource nach einer gewissen Wartezeit abgebrochen und belegte Ressourcen zwischenzeitlich wieder freigegeben werden, bevor diese erneut angefordert werden.

  Man darf Gabeln nicht einzeln aufnehmen, sondern nur die linke und rechte Gabel gleichzeitig.

- Zyklisches Warten kann nicht auftreten, wenn für die Betriebsmittel vorab eine lineare Ordnung festgelegt wird: Ein Betriebsmittel B_j kann nach einem bereits belegten Betriebsmittel B_i nur dann beansprucht werden, wenn $j > i$ gilt.

  Die Gabeln werden durchnummeriert. Ein Philosoph darf seine zweite Gabel nur aufnehmen, wenn ihre Nummer größer als die seiner ersten Gabel ist. (Ansonsten muss er die vorige wieder ablegen.)

- Anstelle einer kategorischen „Ununterbrechbarkeit" kann ein kontrollierter Betriebsmittelentzug von außen erlaubt werden. Dabei muss der aktuelle Zustand der entzogenen Ressource gesichert werden können, damit eine spätere Wiederherstellung für eine Threadfortsetzung möglich ist. Dies ist z. B. für Speicherbereiche oder den Prozessor möglich, allerdings nicht für eine **ununterbrechbare Ressource** wie z. B. einen Drucker:
  Die Unterbrechung geht bei ununterbrechbaren Ressourcen immer mit einem Verlust der bisher geleisteten Arbeit einher – beim Abbruch eines Druckvorgangs wird zumindest die aktuelle Seite nur teilbedruckt.

  Einem auf die zweite Gabel wartenden Philosophen darf die erste Gabel vorübergehend weggenommen werden!

Beim gemeinsamen Zugriff auf mehrere Ressourcen kann es zu einer **Verklemmung** kommen, wenn alle beteiligten Threads zur Ausführung Ressourcen benötigen, die einer der anderen Threads bereits belegt. Der **Betriebsmittelzuteilungsgraph** enthält dann einen Zyklus. Zur Verhinderung kann von vornherein dafür gesorgt werden, dass mindestens eine der vier **Coffman-Bedingungen** für Verklemmungen nicht erfüllt ist: wechselseitiger Ausschluss, Halten und Warten, Ununterbrechbarkeit und zyklisches Warten. Wird einem Thread eine **ununterbrechbare Ressource** entzogen, geht die bis dahin geleistete Arbeit verloren.

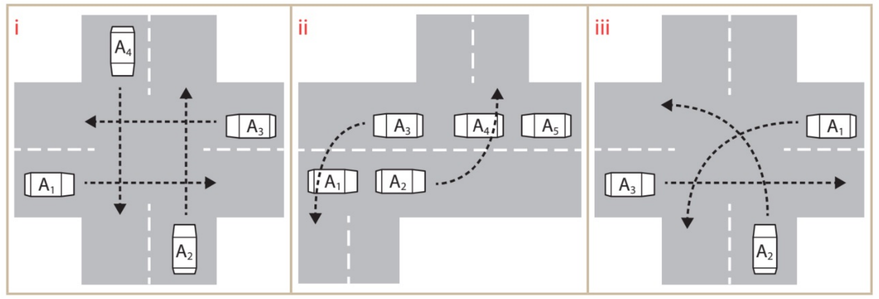
Aufgaben

1 Richtig oder falsch?
Entscheiden Sie zu jeder Aussage, ob sie richtig oder falsch ist. Begründen und berichtigen Sie bei falschen Aussagen.

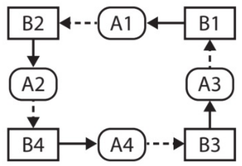
- **a** Damit eine Verklemmung zustande kommen kann, benötigt man mindestens zwei Threads und eine Ressource.
- **b** Eine Verklemmung kann durch gute Synchronisation vermieden werden.
- **c** Beim Erzeuger-Verbraucher-Problem kann es zu einer Verklemmung kommen.
- **d** Zur präventiven Verhinderung von Verklemmungen müssen alle vier Coffman-Bedingungen erfüllt sein.
- **e** Als ununterbrechbare Ressource wird ein Betriebsmittel bezeichnet, das einem Prozess nicht entzogen werden kann.

2 Xylophon
Reflektieren Sie Ihre Lösung von Teilaufgabe c) der Einstiegsaufgabe: Welche Coffman-Bedingung wird bei der von Ihnen vorgeschlagenen Strategie verletzt? Formulieren Sie dann im Sachkontext zwei alternative Regeln, die eine Verklemmung verhindern, indem sie jeweils eine andere Coffman-Bedingung verletzen.

3 Verkehrssituationen
In den vorliegenden Verkehrssituationen wollen die Fahrzeuge in Richtung der Pfeile fahren. Es gelten die Verkehrsregeln „Rechts vor Links" und „Linksabbieger müssen den Gegenverkehr abwarten".



- **a** Erklären Sie, warum es in den dargestellten Verkehrssituationen jeweils zu einer Verklemmung gekommen ist.
- **b** Formulieren Sie jeweils Maßnahmen, die eine solche Verklemmung verhindern würden, ohne die Fahrtmöglichkeiten der Verkehrsteilnehmenden einzuschränken.
- **c** Ordnen Sie den dargestellten Betriebsmittelzuteilungsgraphen einer der Verkehrssituationen zu. Begründen Sie Ihre Zuordnung und erklären Sie, worum es sich bei den Betriebsmitteln B1, B2, B3 und B4 handelt.



4 Campingurlaub

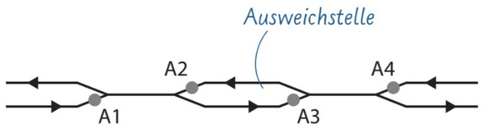
Beim gemeinsamen Kochen im Campingurlaub benötigt Lucia das Sieb zum Salatwaschen und den Mörser für die Kräutersoße. Zumindest das Schälmesser hat sie sich für die Gurken bereits gesichert. Roman nutzt gerade den Topf zum Nudelkochen und will zum Abgießen ebenfalls das Sieb. Für die Gewürzpaste seiner Vorspeise hat er den Mörser in Gebrauch. Giulia bräuchte den Topf für die Soße, außerdem hätte sie gerne den Schäler für die Karotten. Justus nutzt gerade das Sieb, um darin gespültes Essgeschirr abtropfen zu lassen.

a Erklären Sie auf Basis eines Betriebsmittelgraphen, ob die Clique verhungern muss, falls niemand bereit ist, einmal erlangte Küchengeräte vor Erledigung all seiner Aufgaben wieder herzugeben.

b Geben Sie eine „minimalinvasive" Veränderung der Aufgabenstellung an, so dass niemand aus der Gruppe seine Kochaufgabe erledigen kann.

5 Baustelle (aus Abitur 2016)

Eine zweispurige Straße für zwei Richtungen ist im Baustellenbereich auf eine Spur verengt. In der Mitte des Baustellenbereichs wird der einspurige Bereich durch eine zweispurige Ausweichstelle unterbrochen. Die Einfahrt in die einspurigen Teilbereiche wird durch insgesamt vier Ampeln A1 bis A4 gesteuert.



Die Ampel A1 kann erst dann auf grün schalten, wenn A2 rot zeigt; umgekehrt kann A2 erst dann auf grün schalten, wenn A1 rot zeigt. Entsprechendes gilt für A3 und A4. Die Ampeln arbeiten in wechselnden zeitlich getakteten Rot- und Grünphasen, zusätzlich unterstützt durch Sensoren. Dadurch wird sichergestellt, dass eine Ampel nur dann auf grün schaltet, wenn der unmittelbar folgende einspurige Bereich völlig frei ist.

Geben Sie an, was man in der Informatik allgemein unter Verklemmung versteht, und führen Sie an einem Beispiel aus, dass die genannten Regeln der Ampelsteuerung nicht ausreichen, um eine Verklemmungssituation in diesem Straßenabschnitt zu verhindern.

6 Philosophenproblem

a Testen Sie den Ablauf beim Problem der speisenden Philosophen zu zweit: Erstellen Sie im vorliegenden Projekt jeweils ein Objekt der Klasse SPEISENDE_PHILOSOPHEN und stoppen Sie jeweils die Zeit, bis es zu einer Verklemmung kommt. Erklären Sie sich gegenseitig auf Basis der gemessenen Zeiten, warum es sich um eine Race Condition (vgl. Sprechblase) und keinen algorithmischen Fehler handelt. Verkleinern Sie dann für zukünftige Tests das Zeitintervall für die zufallsgesteuerte Essenszeit deutlich, indem Sie die Wartezeit in PHILOSOPH z. B. auf 10 ms verkürzen.

Im Folgenden soll die Verklemmung verhindert werden, indem Coffman-Bedingung 4 („Zyklisches Warten") verletzt wird: Die notwendige lineare Betriebsmittelordnung kann durch die Indizes der Gabeln im Feld ausgedrückt werden. Diese werden bereits als id einer Gabel gespeichert.

Hängt das Ergebnis eines Algorithmus von der nicht vorhersehbaren zeitlichen Reihung der nebenläufigen Einzeloperationen ab, wird dies Race Condition genannt. Diese Laufzeitfehler sind oft besonders schwierig zu entdecken, da sie ohne erkennbares System nur manchmal auftreten.

Das Struktogramm zeigt eine Umsetzung der Strategie:

PhilosophenLineareOrdnung

…
gabelLinks.Aufnehmen(tellerfarbe)
gabelRechts.IdGeben() > gabelLinks.IdGeben() wahr / falsch
gabelRechts.Aufnehmen(tellerfarbe) | gabelLinks.Ablegen() gabelRechts.Aufnehmen(tellerfarbe) gabelLinks.Aufnehmen(tellerfarbe)
…

b Stellen Sie das Philosophenproblem in einer Dreiergruppe nach und setzen Sie dabei die Strategie aus dem Struktogramm um. Erklären Sie dann, für welche(n) Philosophen sich dadurch eine veränderte Vorgehensweise bei der Gabelaufnahme ergibt.

c Setzen Sie die Strategie in der Methode *Arbeiten* um. Beobachten Sie das veränderte Programmverhalten.

In einer Kopie des Projekts soll nun eine weitere Strategie zur Verhinderung der Verklemmung umgesetzt werden, die die Coffman-Bedingung 2 („Halten und Warten") verletzt. Die dazu im Lehrtext formulierte Strategie würde einen größeren Umbau des Projekts erfordern. Daher wird eine andere Variante implementiert, bei welcher der Wartevorgang auf die zweite Gabel nach einer begrenzten Zeitspanne abgebrochen wird. Dazu muss ein Philosoph nach Aufnahme der ersten Gabel für die zweite Gabel eine neue Methode *AufnehmenVersuchen(eigeneFarbe)*: WAHRHEITSWERT aufrufen.

synchronisierte Methode AufnehmenVersuchen(eigeneFarbe): WAHRHEITSWERT

n = 0
solange n < 100 NICHT inBenutzung wahr / falsch
inBenutzung = WAHR FarbeSetzen(eigeneFarbe) return WAHR | Warten(10) n = n + 1
return FALSCH

d Vollziehen Sie zu zweit die Arbeitsweise der neuen Methode anhand des Pseudocodes nach.

e Liefert der Methodenaufruf von *AufnehmenVersuchen* den Wert WAHR zurück, kann der Philosoph essen und legt anschließend die zuletzt aufgenommene Gabel wieder ab; in jedem Fall muss er dann auch die zuerst aufgenommene Gabel wieder ablegen. Setzen Sie die Methode in der Klasse GABEL um und nehmen Sie auch die beschriebenen Anpassungen in der Klasse PHILOSOPH vor. Testen Sie!

Teste dich selbst

T1 Richtig oder falsch?

Beurteilen Sie, ob folgende Aussagen richtig oder falsch sind. Begründen Sie Ihre Meinung bei falschen Aussagen und geben Sie eine berichtigte Aussage an:

- **a** Ein nebenläufiges Programm besteht aus mehreren Ausführungssträngen, die immer parallel ausgeführt werden.
- **b** Ein kritischer Abschnitt darf stets nur von einem Thread gleichzeitig durchlaufen werden.
- **c** Beim aktiven Warten versucht ein Thread permanent, einen geschützten Codeabschnitt zu betreten.
- **d** Bei der gemeinsamen Nutzung eines Speicherplatzes durch einen Erzeuger und einen Verbraucher besteht die Gefahr einer Verklemmung.
- **e** Hat ein Thread *Warten()* aufgerufen und einen Monitor freigegeben, wird er automatisch wieder aktiv, wenn der Monitor wieder frei ist.
- **f** Eine Verklemmung kann verhindert werden, indem man dafür sorgt, dass die vier Coffman-Bedingungen nicht erfüllt sind.

T2 Ich check's, dank deiner Hilfe!

Ferdi hat im Unterricht nicht gut aufgepasst und braucht dringend Ihre Nachhilfe vor der alles entscheidenden letzten Prüfung.

- **a** Helfen Sie ihm, indem Sie ihm die folgenden Begriffe bzw. Konzepte erklären und zudem an einem Beispiel aus Ihrer Programmiersprache zeigen, wie man damit umgeht (am bestem am Rechner, nicht auf Papier!):
  Thread, Monitor, Warten und Benachrichtigen
- **b** Erläutern Sie ihm zusätzlich Strategien, um Verklemmungen zu verhindern.

T3 Schwierigkeiten beim Tabletverleih

Der Wahlkurs Programmieren hat für die Schule ein System entwickelt, mit dem der Verleih von Schultablets verwaltet werden kann. Schülerinnen oder Schüler können bei Bedarf bei einem Mitglied des Verleihteams anfragen; im Verwaltungsprogramm wird dann ein eigener Programmthread auf dem Schulserver gestartet und die passende Methode (*GerätAusleihen(gerät)* oder *GerätZurückgeben(gerät)*) aufgerufen.

In einer ersten Testversion der Software haben die beiden Methoden einer Klasse GERÄTEVERWALTUNG das folgende Aussehen:

methode GerätAusleihen(gerät): WAHRHEITSWERT

anzahlGeräte > 0	
wahr	falsch
anzahlGeräte = anzahlGeräte - 1 gerät.EntleihdatenAbfragen() return WAHR	return FALSCH

methode GerätZurückgeben(gerät)

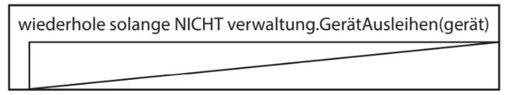
```
anzahlGeräte = anzahlGeräte + 1
gerät.Freigeben()
```


- **a** Begründen Sie unter Angabe eines Beispielszenarios im Sachkontext, warum es hierbei zu Fehlern kommen kann.
- **b** Geben Sie eine Verbesserung für den dargestellten Programmcode an, so dass der Fehler aus a) nicht mehr auftreten kann.

Nach dem Aufruf von *GerätAusleihen* müssen Mitglieder des Verleihteams alle Daten der Schülerinnen bzw. Schüler in eine Eingabemaske eintragen. Erst dann wird die Methode erfolgreich beendet. Gerade zu Schuljahresbeginn, wenn viele Geräte verliehen werden sollen, kann es dabei zu Wartezeiten der Threads kommen.

- **c** Geben Sie an, durch welche Umstrukturierung Sie eine bessere Parallelisierung erreichen könnten.

Ist eine der Methoden in Ausführung, müssen andere Threads aufgrund der Verbesserung in b) warten. Das Struktogramm zeigt die Umsetzung einer Wartestrategie:



- **d** Erklären Sie, welche Wartestrategie umgesetzt wurde, und bewerten Sie diese.
- **e** Geben Sie Pseudocode für eine ressourcenschonendere Wartestrategie an.

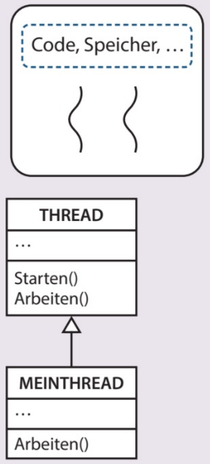
Nach ersten Testläufen und Verbesserungen wurde das Programm erweitert: Zusätzlich zu Tablets sollen nun auch Tabletstifte verliehen werden. Alle gewünschten Geräte werden in einen virtuellen Warenkorb gelegt; Ein Verleihvorgang ist dann abgeschlossen, wenn mit jedem Gerät in der Reihenfolge des Hinzufügens zum Warenkorb die Methode *GerätAusleihen* aufgerufen wurde.

- **f** Beschreiben Sie eine Situation, in der es bei dieser Umsetzung zu einer Verklemmung kommen kann, und illustrieren Sie diese mit einem geeigneten Graphen.
- **g** Nennen Sie erst allgemein Mechanismen, mit denen eine Verklemmung verhindert werden kann, und formulieren Sie dann eine Umsetzungsstrategie im Sachkontext.

Zusammenfassung

Nebenläufige Programme bestehen aus mehreren Ausführungssträngen, welche parallel ausgeführt werden können. Eine einfache Möglichkeit, um dies zu erreichen, ist das Erzeugen mehrerer **Threads** innerhalb eines Prozesses.
Viele Programmiersprachen stellen hierfür eine Klasse THREAD bereit, deren *Arbeiten()*-Methode in einer Unterklasse individuell überschrieben werden kann. Einmal gestartet, werden die Threads unabhängig voneinander ausgeführt. Dies ermöglicht es, rechenintensive Aufgaben auf mehrere parallel arbeitende Rechenkerne zu verteilen. Auch werden Benutzeroberflächen häufig in einen separaten Thread ausgelagert, damit diese auch dann auf Benutzereingaben reagieren können, wenn das Programm im Hintergrund andere Aufgaben ausführt.

ein Prozess mit zwei Threads



Darf ein bestimmter Teilabschnitt eines nebenläufigen Programms nur von einem Thread gleichzeitig durchlaufen werden, wird dies als **kritischer Abschnitt** bezeichnet. Der dann erforderliche **wechselseitige Ausschluss** kann mittels des in einigen Programmiersprachen verfügbaren **Monitorkonzepts** realisiert werden. Dabei werden kritische Abschnitte im Programmcode besonders markiert, die Umsetzung des wechselseitigen Ausschlusses erfolgt dann im Hintergrund.

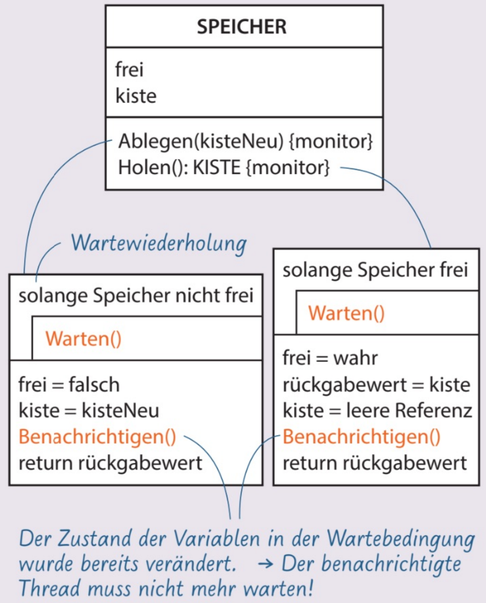
Monitorkonzept:

Java
```
...
synchronized {
   // kritischer Abschnitt
}
```

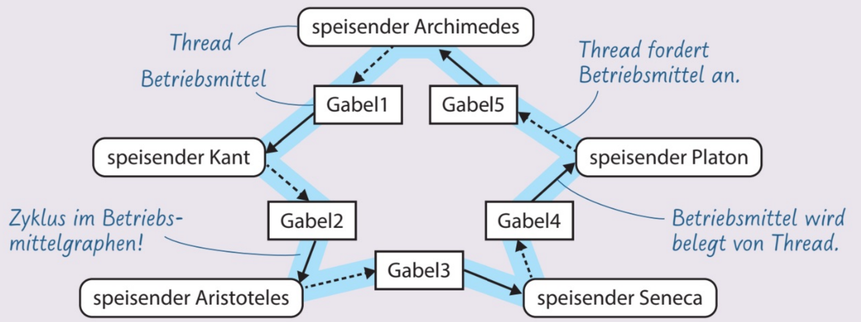
Python
```
import threading
lock = threading.Lock()
...
with lock:
   # kritischer Abschnitt
```

Swift
```
let monitor = NSCondition()
...
monitor.lock()
   // kritischer Abschnitt
monitor.unlock()
```


Das **Erzeuger-Verbraucher-Problem** ist ein Beispiel für ein Synchronisationsproblem, bei dem es vom Zustand der gemeinsam genutzten Ressource abhängt, ob ein Thread seine Aufgabe erledigen kann. Innerhalb eines Monitors kann der Zugriff durch Absprachen koordiniert werden (**passives Warten**): Die Methoden ***Warten*** und ***Benachrichtigen*** (bzw. *AlleBenachrichtigen*) realisieren die Kommunikation zwischen den Threads. So wird eine permanente Neuüberprüfung des Ressourcenzustands (**aktives Warten**) vermieden.



Beim gemeinsamen Zugriff auf mehrere Ressourcen kann es zu einer **Verklemmung** kommen, wenn alle beteiligten Threads zur Ausführung Ressourcen benötigen, die einer der anderen Threads bereits belegt. Der **Betriebsmittelzuteilungsgraph** enthält dann einen Zyklus. Zur Verhinderung kann von vornherein dafür gesorgt werden, dass mindestens eine der vier **Coffman-Bedingungen** für Verklemmungen nicht erfüllt ist: wechselseitiger Ausschluss, Halten und Warten, Ununterbrechbarkeit und Zyklisches Warten. Wird einem Thread eine **ununterbrechbare Ressource** entzogen, geht die bis dahin geleistete Arbeit verloren.



Zum Weiterlesen

L6 Verschiedene Formen der Parallelität

Startet man ein nebenläufiges Programm auf einem Rechner, welcher nur über einen Rechenkern verfügt, so müssen sich die verschiedenen Ausführungsstränge (Threads) den Rechenkern „teilen", indem sie vom Scheduler nacheinander Rechenzeit zugewiesen bekommen. Man spricht in diesem Fall von Pseudoparallelität, da die Befehlsfolgen der einzelnen Threads nicht wirklich parallel ausgeführt werden. Die zugrundeliegende Hardwarearchitektur wird fachsprachlich auch als **Single Instruction, Single Data (SISD)** bezeichnet, da zu jedem Zeitpunkt nur eine Recheninstruktion auf einen Datenwert angewendet werden kann. Für Spezialanwendungen insbesondere im Bereich der Raumfahrt und der kritischen Infrastruktur, wo Fehler katastrophale Folgen haben können, kommen vereinzelt auch Architekturen nach dem Prinzip **Multiple Instruction, Single Data (MISD)** zum Einsatz, bei der die Verarbeitung eines Datenwertes gleich mehrfach von verschiedenen Rechnern durchgeführt wird. Hieraus ergibt sich zwar kein Geschwindigkeitsvorteil; ein Abgleich der Ergebnisse erlaubt dafür das Erkennen hardwarebedingter Rechenfehler.

Während MISD-Systeme im Alltag kaum eine Rolle spielen, wird das Prinzip **Single Instruction, Multiple Data (SIMD)** hingegen heutzutage von einer Vielzahl von Prozessoren unterstützt. Insbesondere bei der Verarbeitung von Grafikdaten kommt es häufig vor, dass eine Rechenoperation auf viele Datenwerte gleichzeitig angewendet werden muss, etwa um sämtliche Pixeldaten eines Bildes auf die gleiche Weise zu verändern. Prozessoren für derartige Anwendungen wurden deshalb bereits früh um entsprechende Instruktionen erweitert, die Verarbeitungsoperationen in einem Schritt auf gleich mehrere Datenwerte anwenden können. Auch die Prozessoren in Grafikkarten unterscheiden sich von den Prozessoren, welche als Hauptprozessor (CPU) in Computern eingesetzt werden, in erster Linie darin, dass ihre Architektur auf die parallele Anwendung von einfachen Rechenoperationen auf eine Vielzahl von Werten spezialisiert ist.

Neben diesen SIMD-Erweiterungen sind heute auch im Privatbereich Computer mit mehreren Rechenkernen Standard. Es können also je nach Anzahl der Rechenkerne mehrere Recheninstruktionen echt parallel ausgeführt werden. Da die Rechenkerne in der Regel zusätzlich die oben genannten SIMD-Befehle beherrschen, spricht man nun von **Multiple Instruction, Multiple Data (MIMD)**. Moderne Supercomputer treiben dieses Prinzip oft auf die Spitze, indem sie als Zusammenschluss von tausenden Einzelrechnern (sog. Computercluster) mit jeweils mehreren Rechenkernen aufgebaut sind.

L7 Kooperative Threads

Der Grundansatz kooperativer (nicht-präemptiver) Threads ist, dass jeder Thread so lange arbeitet, bis er seine Aufgabe erledigt hat, und dann bestimmt, welcher Thread als nächster weiterarbeitet.

Obwohl es bei diesem Konzept nicht möglich ist, bei mehreren verfügbaren Prozessoren Threads parallel ablaufen zu lassen, ist dieses Konzept für viele Modellierungssituationen gut geeignet, weil es bei diesem Konzept keine Synchronisierungsprobleme gibt und weil beim Debuggen alle Situationen einfach reproduzierbar sind. Auch benötigen kooperative Threads keinerlei Unterstützung des Betriebssystems.

Ein typisches Beispiel für kooperative Threads ist in der Programmiersprache Modula-2 realisiert; die Threads werden hier als Koroutinen bezeichnet, um diese auch vom Namen her von präemptiven Threads abzugrenzen. Als Beispiel für die Verwendung dieses Konzepts dient hier eine Mehrspielerversion eines Fantasy-Abenteuer-Spiels.

Vorhandene Sprachmittel

COROUTINE	Datentyp für die Referenz auf eine Koroutine
NEWCOROUTINE (	Methode zum Erzeugen einer neuen Koroutine
procBody: PROC;	parameterlose Rahmenmethode
workspace: ADDRESS;	niedrigste Adresse des vorgesehenen Stackbereichs
size: CARDINAL;	Größe des Stackbereichs
VAR cr: COROUTINE;	In die hier übergebene Variable wird die Referenz auf die neue Koroutine zurückgegeben.
initprotection: PROTECTION);	Prioritätsangabe
TRANSFER (	wechselt zur einer anderen Koroutine
VAR from: COROUTINE	In die hier übergebene Variable wird die Referenz auf die aktuelle Koroutine zurückgegeben.
to: COROUTINE);	Referenz auf die Koroutine, die jetzt ausgeführt werden soll
COROUTINEDONE (	gibt Verwaltungsressourcen der Koroutine frei
cr: COROUTINE);	freizugebende Koroutine
CURRENT (): COROUTINE;	meldet die aktuelle Koroutine

Umsetzung im Abenteuerspiel

Die Nutzung der Koroutinen für das Spiel ist einfach:

- Jedem Spieler wird eine eigene Koroutine zugeordnet, so dass jeder Spieler seinen eigenen, von anderen abgegrenzten Zustand verwaltet.
- Jede Spielerkoroutine gibt den Prozessor an die Hauptkoroutine ab, wenn sie auf eine Eingabe warten muss.
- Die Hauptkoroutine richtet zunächst die Spielerkoroutinen ein und startet sie. Dann holt sie eine Eingabe nach der anderen und gibt den Prozessor an die der Eingabe zugeordnete Koroutine ab. Diese Wiederholung wird erst beendet, wenn ein Spieler das Spielziel erreicht hat.

Durch diesen Mechanismus kann jeder Spieler die Kontrolle an einer beliebig tiefen Aufrufverschachtelung abgeben und an dieser Stelle wieder fortsetzen. Alle Aktionen zwischen zwei TRANSFER-Aufrufen werden garantiert an einem Stück ausgeführt. Daher müssen keinerlei Vorkehrungen getroffen werden, um zwischenzeitlich inkonsistente Datenzustände vor anderen Zugriffen abzusichern.

Anlegen des Stackbereichs und der Koroutinen für jeden Spieler
Die Spieler sind in einer verketteten Liste verwaltet. Diese Liste wird für alle Spieler durchlaufen.

```
PROCEDURE SpielerRoutinenAnlegen;
BEGIN
   aktSpieler := erster;
   WHILE aktSpieler <> EMPTY DO
      (* Anlegen des Stackbereichs *)
      NEW (stackPointer[aktSpieler.spielerNummer]);
      (* Anlegen der Koroutine *)
      NEWCOROUTINE (SpielerProcess,
          stackPointer [aktSpieler.spielerNummer],
          stackGröße, aktSpieler.spielerProc);
      (* Start der Koroutine *)
      TRANSFER (mainProc, aktSpieler.spielerProc);
      (* Nach der Initialisierung wartet die Spielerkoroutine
         Auf die erste Eingabe und gibt daher den
         Prozessor wieder an die Hauptkoroutine ab.
       *)
       aktSpieler := aktSpieler.naechster;
   END(*WHILE*);
END SpielerRoutinenAnlegen;
```

Eingabeaufforderung eines Spielers für eine ganze Zahl
Für jede Art der gewünschten Eingabe ruft der Spieler eine passende Methode auf. Der Eingabedialog wird nur am Computer des aktuellen Spielers angezeigt. Die Eingabe wird über das Netzwerk an alle Mitspieler verteilt, so dass alle beteiligten Programm synchron laufen.

```
PROCEDURE GetZahl (VAR param: CARDINAL; text: ARRAY OF CHAR);
BEGIN
    (* Der Eingabedialog wird gezeigt *)
    GetNumDialog(text);
    (* Der Spieler wird als "wartend auf Eingabe" markiert *)
    INCL (aktSpieler. spielerSignale, eaWartend);
    (* Prozessor wird an die Hauptroutine abgegeben *)
    TRANSFER (aktSpieler. spielerProc, mainProc);
    (* Eine Eingabe für den Spieler ist gekommen.
       Daher gibt die Hauptkoroutine den Prozessor wieder an
       Die Spielerkoroutine ab.
    *)
    param := eingabe[aktSpieler.spielerNummer].nummer;
END GetZahl;
```

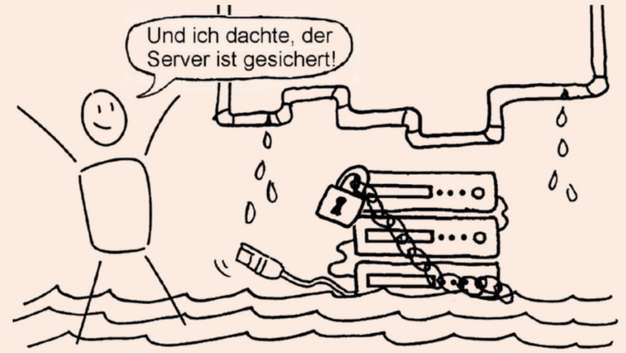
Betriebssystem in Modula-2
Das Koroutionenkonzept von Modula-2 enthält zusätzliche Methoden, die es ermöglichen, dass auch durch externe Ereignisse (Interrupts) zu einer Koroutine transferiert wird. Dadurch ist es möglich, auch Betriebssysteme in Modula-2 zu schreiben.
Das bekannteste Beispiel ist das Betriebssystem Medos-2, das auf der an der ETH Zürich entwickelten Grafikworkstation Lilith eingesetzt wurde.

5 Informationssicherheit

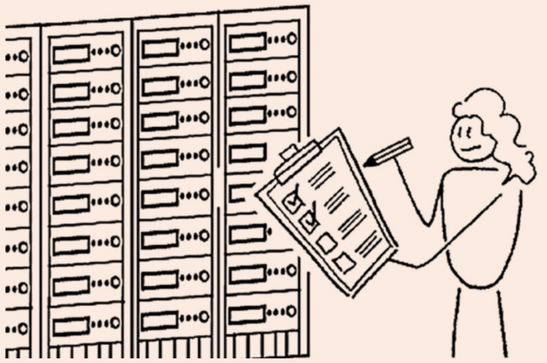
In diesem Kapitel werden Sie ...

... die wichtigsten Schutzziele der Informationssicherheit kennenlernen.

... die wichtigsten Maßnahmen kennenlernen, um IT-Systeme zu schützen.



... IT-Systeme hinsichtlich ihrer Sicherheit analysieren.



5.1 Schwachstellen analysieren: Gefahren für IT-Systeme

Insolvenz wegen Serverausfalls

Nach fast 60 Jahren muss ein mittelständischer Küchenhersteller Insolvenz anmelden. Grund für die Insolvenz ist ein vorausgegangener Totalausfall der Server. Dies führte zu solch großen Datenverlusten, dass die Produktion in der Folge gestoppt werden musste. Eine Wiederaufnahme war nicht mehr möglich.

a Finden Sie durch eine Recherche heraus, ob der Zeitungsartikel auf einer wahren Begebenheit beruht und ob solche Vorfälle öfter auftreten.
b Eine mögliche Ursache für den Serverausfall könnte ein Hackerangriff gewesen sein. Recherchieren Sie die Höhe der Schadenssumme, welche der deutschen Wirtschaft im letzten Jahr durch IT-Angriffe entstanden ist.
c Sammeln Sie weitere Szenarien, die zum Serverausfall geführt haben könnten.
d Ergänzen Sie die Sammlung aus c) um weitere Gefahren für IT-Systeme. Finden Sie Oberbegriffe und ordnen Sie diesen die Szenarien und Gefahren zu.
e Für Schnelle: Überlegen Sie, welche persönlichen und psychologischen Konsequenzen das im Zeitungsartikel beschriebene Szenario für die Mitarbeiter mit sich bringen kann.

Gefahren für IT-Systeme

Jährlich entsteht der deutschen Wirtschaft ein Schaden in Höhe von mehreren Milliarden Euro durch Angriffe auf IT-Systeme. Teilweise geraten Unternehmen sogar an den Rand der Insolvenz. Dabei ist ein Hackerangriff längst nicht die einzige Gefahr für IT-Systeme. Auch ein Stromausfall oder ein kleiner Bedienfehler können fatale Folgen haben. Um ein IT-System zu schützen, gilt es zunächst, die möglichen Gefahren und Angriffspunkte zu identifizieren, um dann geeignete Maßnahmen zu ergreifen. Die folgende Mindmap stellt mögliche Gefährdungen übersichtlich dar.

Die Bandbreite möglicher Gefahren ist immens, unterscheidet sich aber auch je nach System. Für ein im Krankenhaus eingesetztes System, das die Beatmung der Patientinnen und Patienten sicherstellt, gibt es andere Gefahren als für eine Messenger-Anwendung.

Schutzziele

Aufgrund der enormen Bedeutung von Informatiksystemen und Daten gilt es, deren Verfügbarkeit zu garantieren und Missbrauch zu verhindern. Um die Gefahren zu minimieren, versuchen die Betreiber von IT-Systemen die vier Schutzziele **Vertraulichkeit**, **Integrität**, **Verfügbarkeit** und **Authentizität** zu erreichen.

- Vertraulichkeit
  Egal ob Nutzerdaten eines sozialen Netzwerks, relevante Ergebnisse eines streng geheimen Forschungsprogramms oder Akten einer Anwaltskanzlei: Meistens gibt es Daten und Dienste, die nur für ausgewählte Personen (auch innerhalb eines Unternehmens) bestimmt sind. Vertraulichkeit ist dann gegeben, wenn nur berechtigte Personen Zugriff auf bestimmte Daten und Dienste haben.

- Integrität
  IT-Systeme verwalten viele wichtige Informationen wie Kontostände, Lagerbestände oder Schulnoten. Natürlich sollte niemand in der Lage sein, diese Daten heimlich zu verändern. Integrität ist gewährleistet, wenn die auf IT-Systemen gespeicherten Daten nicht unbemerkt verändert werden können, also die Korrektheit der Daten und damit die korrekte Funktionsweise von IT-Systemen gewährleistet ist.

- Verfügbarkeit
  Fällt eine Anwendung aus, die zur Beatmung schwerkranker Patientinnen und Patienten eingesetzt wird, könnte das schwere gesundheitliche Schäden zur Folge haben. Daher ist es wichtig, die Verfügbarkeit von IT-Systemen so sicherzustellen, dass Daten oder Dienste dann abgerufen werden können, wenn sie benötigt werden. Verfügbarkeit impliziert jedoch nicht, dass alle IT-Systeme zu jeder Zeit verfügbar sein müssen. Vielmehr ergeben sich je nach Anwendungsfall verschiedene Anforderungen an die Verfügbarkeit: Das Bestellsystem eines Restaurants muss lediglich während der Geschäftszeiten verfügbar sein, die lebenserhaltenden Systeme eines Krankenhauses müssen hingegen zu jeder Zeit verfügbar und vor Ausfällen geschützt sein.

- Authentizität
  Gerade im Internet ist sicherzustellen, dass man z. B. als Kunde mit der „echten" Bank kommuniziert. Phishing-Webseiten versuchen sich häufig als Webseiten von Banken oder anderen Firmen auszugeben, um so an Daten zu gelangen und Schaden anzurichten. Auch bei Messenger-Diensten sollte niemand in der Lage sein, sich einfach als jemand anderes auszugeben. Damit Authentizität gewährleistet ist, müssen Nutzende in der Lage sein, den Ursprung bzw. die Echtheit von Daten oder von Diensten zu verifizieren bzw. zu prüfen.

Gezielte Angriffe, **technisches Versagen**, **organisatorische Mängel** oder **menschliche Fehlhandlungen** stellen eine enorme Gefahr für IT-Systeme dar. Neben großen wirtschaftlichen Schäden können auch persönliche Schäden (z. B. Verlust des Arbeitsplatzes) entstehen. Daher ist es wichtig, diese Systeme zu schützen und so **Vertraulichkeit**, **Integrität**, **Verfügbarkeit** und **Authentizität** zu gewährleisten.

Aufgaben

1 Opfer von IT-Sicherheitsvorfällen

Kennen Sie jemanden in Ihrem Freundes- und Bekanntenkreis, der bereits von einem IT-Sicherheitsvorfall betroffen war (oder waren Sie vielleicht persönlich betroffen)? Falls nicht, recherchieren Sie einen Vorfall, bei dem Privatpersonen betroffen waren, und beantworten Sie die Fragen anhand dieses Vorfalls. Bewahren Sie dabei die Anonymität der betroffenen Person.

a Ordnen Sie den Vorfall der Mindmap auf S. 148 zu.
b Ordnen Sie die Folgen auf einer Skala von 1–6 zu (6 bedeutet fatale Folgen) und erläutern Sie die Zuordnung.
c Reflektieren Sie gemeinsam, welche Schutzmaßnahmen den Vorfall verhindern hätten können.

2 Gefährdungslage in Deutschland

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) veröffentlicht jährlich einen Bericht zur Lage der IT-Sicherheit in Deutschland. Analysieren und vergleichen Sie den aktuellsten Bericht mit einem mindestens fünf Jahre alten Bericht. Folgende Aufgabenstellungen helfen dabei:

a Geben Sie die Gefahren an, welche der aktuelle Bericht auflistet, und ordnen Sie diese den Kategorien der Mindmap auf S. 148 zu.
b Finden Sie heraus, welche Gefahren im aktuellen und älteren Bericht genannt werden, welche nur in der Vergangenheit eine Rolle spielten und welche neu im Vergleich zum alten Bericht sind.
c Der Bericht nennt konkrete Vorfälle. Analysieren Sie einen Vorfall hinsichtlich der Erreichung bzw. Nicht-Erreichung der vier Schutzziele.

3 Phishing-E-Mails

Mit Phishing-E-Mails versuchen Cyberkriminelle ihre Opfer auf eine fingierte Webseite zu locken und zur Eingabe sensibler Informationen zu bewegen. Dabei greifen Angreifende in die psychologische Trickkiste und spielen bewusst mit menschlichen Emotionen, wenn in Phishing-E-Mails Dringlichkeit suggeriert oder eine Drohkulisse möglicher negativer Konsequenzen aufgebaut wird.

> Von: Jakob <jakob@statd.infohausen.de>
> An: stefan@stadt.infohausen.de
>
> Hallo Stefan,
>
> könntest du noch schnell die folgende Präsentation freigeben:
> http://www.statd.infohausen.de
> Brauche sie dringend (!) für ein Kundengespräch!
>
> Jakob

a Bei der in dieser Aufgabe dargestellten E-Mail handelt es sich um eine Phishing-Nachricht. Geben Sie zwei Gründe an, warum diese E-Mail verdächtig ist.
b Recherchieren Sie nach Beispielen von Phishing-E-Mails und notieren Sie sich übliche Tricks, die Ihnen in diesen E-Mails begegnen.
c Sie sollen die Mitarbeiterinnen und Mitarbeiter der Infohausener Maschinen GmbH im Rahmen eines Security-Awareness-Programms für Phishing-E-Mails sensibilisieren. Versetzen Sie sich dazu in die Rolle eines Cyberkriminellen und entwickeln Sie selbst eine Phishing-E-Mail.

4 Gefahren für IT-Systeme

Als Betreiberin bzw. Betreiber eines IT-Systems ist eine regelmäßige Gefahrenanalyse wichtiger Teil der Sicherheitsstrategie. Wählen Sie eines der gegebenen IT-Systeme aus:

- Smartphone-App, die Zugriff auf den Vertretungsplan ermöglicht
- Börse zum An- und Verkauf von Kryptowährungen
- Smart-Home-System, das Türen, Fenster, Heizungen und Lichter steuert
- Web-Mail-Anbieter
- Internetknotenpunkt
- System zur Verwaltung von Kundenkontakten und Bestellungen
- Versionsverwaltungssystem eines Videospieleherstellers

a Analysieren Sie das gewählte IT-System hinsichtlich möglicher Schwachstellen und Gefahren.
b Priorisieren Sie die Bedeutung der vier Schutzziele in Bezug auf Ihr gewähltes IT-System. Recherchieren Sie dazu gegebenenfalls.
c Skizzieren Sie die möglichen Folgen beim Eintreten eines Unglücksfalls.
d Erstellen Sie eine kurze Präsentation und stellen Sie Ihre Untersuchungsergebnisse anschließend dem Kurs vor.

5 Schutzziele in der Schule

a In den IT-Systemen der Schule werden viele vertrauliche und personenbezogene Daten von Schülerinnen und Schülern, Eltern und Lehrkräften gespeichert. Geben Sie für jedes Schutzziel ein Beispielszenario an, das die Wichtigkeit dieses Ziels verdeutlicht. Gehen Sie dabei insbesondere auf den Datenschutz ein.
b Erläutern Sie, warum sich Lehrkräfte Gedanken über das Erreichen der Schutzziele auf privaten wie dienstlichen Geräten machen sollten.

6 Analyse verschiedener Angriffsszenarien

Beschreiben Sie den Ablauf folgender Angriffsszenarien arbeitsteilig. Recherchieren Sie dazu ggf. im Internet. Stellen Sie Ihre Ergebnisse anschließend der Gruppe vor. Gehen Sie dabei auf die „Funktionsweise" des Szenarios ein und geben Sie ein Beispiel für einen konkreten Vorfall an. Erläutern Sie zudem, welche Auswirkungen ein Angriff haben kann.

a Schadsoftware wie Viren, Trojanische Pferde (Trojaner)
b Ransomware
c Drive-by-Exploits
d Botnetze
e Identitätsdiebstahl
f Denial-of-Service (DoS) bzw. Distributed Denial-of-Service (DDoS)
g Advanced Persistent Threats
h Kennwortangriffe
i Man-in-the-Middle-Attacke
j Social Engineering

7 Forschungsauftrag: Websites hacken

Um die eigenen Abwehrfähigkeiten zu üben, gibt es im Internet verschiedene Cybersecurity-, sog. „Capture the Flag"-Übungen. Suchen Sie nach solchen Aufgaben und versuchen Sie diese zu lösen.

5.2 Systeme schützen: Maßnahmen der IT-Sicherheit

Die Insolvenz des mittelständischen Küchenherstellers durch den Serverausfall (s. Einstiegsaufgabe vorheriges Kapitel) war vermeidbar. Medienberichten zufolge war der Crash auf einen Stromausfall zurückzuführen.

a Sammeln und diskutieren Sie Ideen, welche Vorkehrungen der Hersteller hätte treffen können, um den Serverausfall zu verhindern.

b Nennen Sie Maßnahmen, welche – im Vorfeld getroffen – geholfen hätten, die Systeme nach dem Ausfall wieder instand zu setzen.

c Ein Angriffstyp, mit dem viele Unternehmen zu kämpfen haben, ist ein Ransomware-Angriff. Dabei wird versucht, von der angegriffenen Firma ein Lösegeld für die Daten, die von den Kriminellen verschlüsselt worden sind, zu erpressen. Geben Sie Maßnahmen an, welche diesen Angriff verhindern könnten. Recherchieren Sie ggf. im Internet.

d Erläutern Sie, ob die zu b) notierten Maßnahmen auch bei Ransomware hilfreich sind.

Schutzziele erreichen – Maßnahmen ergreifen

Um die Gefahren für IT-Systeme zu minimieren und die Schutzziele zu erreichen, sollten verschiedene Maßnahmen getroffen werden. Man unterscheidet dabei zwischen **technischen, physischen** und **organisatorischen Maßnahmen**.

Technische Maßnahmen

Bei vielen gezielten Angriffen wird versucht, Schwachstellen in der Software aufzuspüren und auszunutzen. Die dadurch erreichten Auswirkungen technischen Versagens lassen sich durch geeignete Maßnahmen minimieren: **Antivirensoftware** und **Firewall** sind obligatorisch, aber immer noch nicht selbstverständlich. Spamfilter reduzieren die Gefahr, dass Schadsoftware über E-Mails und unachtsame Nutzer in das System gelangen kann. Die **Sandbox** stellt ein weiteres Konzept zum Schutz gegen Schadsoftware dar: Sie ist eine abgeschirmte Umgebung, in der Software getestet werden kann, ohne dass es Auswirkungen auf das System gibt.
Verschlüsselung erschwert es Angreifenden, an vertrauliche Informationen zu gelangen. Durch regelmäßige Daten- und Systemsicherungen (**Backups**) kann ein System bei irreparablen Schäden wieder in den letzten funktionierenden Zustand gebracht werden. Auch die Dokumentation von Zugriffen (z. B. über Logfiles) kann helfen, einen Vorfall im Nachgang besser aufzuarbeiten. Ausgiebiges Testen von neuer Soft- und Hardware (z. B. USB-Stick-Checker) ist eine weitere Maßnahme, die präventiv getroffen werden kann. Wichtig ist zudem, das verwendete Betriebssystem und die Software stets auf dem aktuellen Stand zu halten (Updates), um behobene Sicherheitslücken zu schließen.

Physische Maßnahmen

Kriminelle versuchen aber nicht nur über Lücken in der Software auf IT-Systeme zuzugreifen, Angriffe können auch mit einem Einbruch in die Gebäude beginnen. Daneben können Elementarereignisse wie Starkregen oder Erdbeben, eindringender Staub oder Überhitzung weitere Gefahren darstellen. Physische Maßnahmen sollen IT-Systeme sowohl vor dem Zugriff fremder Personen als auch vor Umwelteinflüssen wie eindringendem Wasser oder Schadstoffen schützen. Bauliche Maßnahmen wie Kühlung, Staubfilter oder allgemeine Brandschutzvorkehrungen verringern die Gefahren von Umwelteinflüssen. Die Unterbringung von Geräten in abgeschlossenen Räumen bzw. Schränken und die Nutzung von Zutrittskontrollsystemen kann zudem auch ein Schutz gegen unerwünschte Eindringlinge oder Vandalismus sein. Zusätzliche Überwachungsmaßnahmen (z. B. Kameraüberwachung) können die Sicherheit erhöhen, stehen aber immer im

Spannungsfeld mit dem Datenschutz. Vor technischem Versagen wie einem Stromausfall können beispielsweise Notstromaggregate schützen. Nach einem **Hardwareausfall** können Backup-Systeme den Zustand vor dem Ausfall wiederherstellen.

Organisatorische Maßnahmen

Viele Gefahren lassen sich zudem auch durch organisatorische Maßnahmen reduzieren. **Richtlinien** für mobiles Arbeiten oder für den Umgang mit Gästen im Büro und **Passwortregeln** sind wichtige Bestandteile eines Sicherheitskonzepts. Ein Notfallplan (z. B. bei höherer Gewalt) hilft im Fall der Fälle, schnell reagieren zu können.
Auch die regelmäßige Überprüfung der räumlichen Gegebenheiten (etwa der Bausubstanz, Feuerlöscher etc.), aber auch der technischen Maßnahmen (z. B. durch → Pentests) sollte Teil der organisatorischen Maßnahmen sein. Um die Mitarbeiterinnen und Mitarbeiter in die Sicherheitsstrategie mit einzubeziehen, sie für diese Gefahren zu sensibilisieren und über notwendige Aktionen (z. B. Meldung verdächtiger Aktivitäten) zu informieren, helfen kontinuierliche Trainings und Weiterbildungen, auch Security-Awareness-Programme genannt.

→ Kurzform von Penetrationstest: intensive Sicherheitstests eines Systems, um Sicherheitslücken aufzudecken

Der Traum des vollkommenen Schutzes

Bei allen ergriffenen Maßnahmen muss ein Unternehmen stets Aufwand und Kosten der Sicherheitsmaßnahmen (z. B. regelmäßige Pentests und Security-Awareness-Schulungen) gegen die Risiken abwägen. Auch wenn alle Sicherheitsmaßnahmen in einem Unternehmen umgesetzt werden, wird ein vollständiger Schutz der IT-Systeme nie erreichbar sein. Bis eine gerade erst entdeckte Sicherheitslücke (eine sog. Zero-Day-Lücke) behoben ist, haben Kriminelle genügend Zeit, die Lücke auszunutzen. Eine 100-%-Sicherheit wird man auch deshalb nie erreichen, da mit Social Engineering die Schwachstelle Mensch ausgenutzt werden kann.

Zum Schutz von IT-Systemen können verschiedene Maßnahmen getroffen werden. Man unterscheidet dabei **physische** Maßnahmen (z. B. Zugangskontrolle), **technische** Maßnahmen (z. B. Virensoftware, Verschlüsselung, Backups) und **organisatorische** Maßnahmen (z. B. Richtlinien, Passwortregeln).
Ein absolut sicheres System kann und wird es aus technischen, wirtschaftlichen und menschlichen Gründen nicht geben.

Aufgaben

1 Maßnahmen gegen verschiedene Angriffsszenarien

Erarbeiten Sie arbeitsteilig, welche Maßnahmen gegen die Angriffsszenarien aus Aufgabe 6 in Kapitel 5.1 getroffen werden können. Stellen Sie Ihre Ergebnisse anschließend dem Kurs vor.

2 Passwortmanager

a Informieren Sie sich über Passwortmanager.

b Beschreiben Sie Vor- und Nachteile von Passwortmanagern.

c Erläutern Sie andere Möglichkeiten, um sichere Passwörter zu generieren und sich diese zu merken.

d Überprüfen Sie Ihre Passwörter hinsichtlich der Sicherheit und passen Sie diese an. Verwenden Sie ggf. einen Passwortmanager.

3 Ransomware und mehr – Geld her!

a Die bekannteste Form von Schadsoftware ist Ransomware, die gespeicherte Daten unleserlich macht. Die Angreifenden fordern dann Lösegeld, damit die Daten wieder „entschlüsselt" werden. Recherchieren Sie mindestens zwei solcher Ransomware-Angriffe. Beschreiben Sie den Ablauf und die Auswirkungen der Angriffe.

b Nennen Sie technische und organisatorische Maßnahmen gegen Ransomware.

c Kriminelle erpressen die Opfer auch mit Schweige- und Schutzgeld. Erklären Sie, für welches Schweigen und für welchen Schutz bezahlt werden soll.

d Diskutieren Sie, ob es sinnvoll ist, Geld an kriminelle Hacker zu zahlen. Beziehen Sie sich in Ihrer Diskussion auf Ihre recherchierten Ransomware-Angriffe aus Aufgabe a).

4 Schutzmechanismen im digitalen Alltag

Beantworten Sie die folgenden Fragen ehrlich und reflektieren Sie am Ende, ob Sie den Schutz Ihrer persönlichen Daten erhöhen können. Sollten Sie auf eine der Fragen keine Antwort wissen, überprüfen bzw. recherchieren Sie, bis Sie die Antwort haben.

a Haben Sie auf Ihrem Handy einen Virenscanner installiert?

b Gibt es in Ihrem Heimnetzwerk eine Firewall?

c Führen Sie Softwareaktualisierungen unmittelbar und regelmäßig durch?

d Verwendet der von Ihnen genutzte Messenger eine Ende-zu-Ende-Verschlüsselung?

e Beträgt die Standardlänge Ihrer Kennwörter über zehn Zeichen und enthalten sie Sonderzeichen?

f Verwenden Sie ein Kennwort mehrfach?

g Machen Sie regelmäßige Backups Ihrer Daten?

h Haben Sie in Ihrem Browser und Messenger Ihre Datenschutzeinstellungen manuell überprüft und ggf. angepasst?

5 Umgang mit Sicherheitslücken

Links zu weiteren Videos zu diesem Thema finden sich im Downloadangebot.

Die Veröffentlichung von Schwachstellen kann eine Maßnahme sein, um die IT-Sicherheit zu erhöhen. Fachleute können aus den gefundenen Fehlern lernen und ggf. gemeinsam Lösungen dafür diskutieren. Dabei ist es wichtig, dass die Personen, welche Lücken finden (z. B. Sicherheitsforscher oder sog. Whitehacker), verantwortungsvoll mit der Aufdeckung umgehen.

a Im Mai 2021 hat die Sicherheitsforscherin Lilith Wittmann in einer App der CDU gravierende Sicherheitslücken gefunden, welche sie der CDU meldete. Daraufhin erstattete die CDU Anzeige gegen Lilith Wittmann. Informieren Sie sich über den sogenannten Hackerparagraphen. Diskutieren Sie auf Grundlage des Paragraphen, ob die Anzeige gegen Lilith Wittmann rechtens war und welches Strafmaß sie ggf. erwarten könnte.

b Wird das Unternehmen benachrichtigt, damit es Zeit hat, die Lücke vor der Veröffentlichung zu schließen, spricht man von Responsible Disclosure. Im Gegensatz dazu wird bei Full Disclosure eine Schwachstelle sofort an die Öffentlichkeit kommuniziert. Geben Sie an, welches Verfahren Lilith Wittmann gewählt hat.

c Ein Grundsatz der Hackerethik des Chaos Computer Clubs lautet „öffentliche Daten nützen, private Daten schützen". Erklären Sie, inwiefern dieser Grundsatz beim Veröffentlichen einer Schwachstelle mittels Full Disclosure verletzt werden könnte.

d Ebenfalls im Jahr 2021 entdeckte ein Programmierer ein Datenleck bei der Firma Modern Solution. Nachdem er die Firma per E-Mail und Telefon kontaktiert hatte, stritt diese die Lücke ab und reagierte nicht mehr auf weitere Anfragen. Daher veröffentlichte der Programmierer die Schwachstelle. Daraufhin durchsuchte die Polizei seine Wohnung und beschlagnahmte unter anderem sein Mobiltelefon. Beziehen Sie Stellung zu diesem Vorfall und diskutieren Sie die Rechtmäßigkeit.

e Diskutieren Sie in Dreiergruppen mit verschiedenen Rollen (Anbieter von IT-Sicherheitslösungen, Sicherheitsforscher, Datenschützer), ob Sicherheitslücken offengelegt werden sollten oder nicht.

6 Darf der Staat hacken?

Nicht nur Kriminelle hacken sich in fremde Rechner, auch der Staat hat Möglichkeiten, im In- und Ausland als Hacker tätig zu werden.

a Recherchieren Sie, was man unter den Begriffen „Online-Durchsuchung" und „Telekommunikationsüberwachung" versteht. Finden Sie insbesondere heraus, welchen rechtlichen Rahmen es dafür gibt.

b Nach Terroranschlägen werden immer wieder Stimmen in der Politik laut, die fordern, dass der Staat auch verschlüsselte Messenger (z. B. Signal, Threema, WhatsApp) mitlesen darf. Neben der Terrorbekämpfung wird als Argument auch der Kampf gegen Kinderpornografie angeführt. Diskutieren Sie dazu in der Gruppe.

c Fremde Nachrichtendienste hacken sich teilweise in deutsche Behörden, Unternehmen oder Organisationen. Auch der Bundesnachrichtendienst (BND) beschäftigt Hacker, um Informationen über andere Länder, Zielpersonen und bestimmte Organisationen einzuholen. Diskutiert wird dabei immer wieder, ob dem BND auch ein sogenannter Hackback erlaubt sein soll, also eine Art digitaler Vergeltungsschlag. Diskutieren und bilden Sie sich eine Meinung zu einem solchen Angriff. Recherchieren Sie anschließend, ob der Gesetzgeber einen Hackback erlaubt.

→ L8

7 Die menschliche Firewall

Organisatorische Maßnahmen wie Security-Awareness-Schulungen oder die Herausgabe von Leitlinien (etwa für das Verhalten bei der Betreuung von Gästen) und Passwortregeln sind ein wichtiger Bestandteil von IT-Sicherheitsmaßnahmen. Entwickeln Sie im Team ein Poster für eines der unten genannten Themen, das für eine Security-Awareness-Kampagne an Ihrer Schule verwendet werden kann. Gerne dürfen Sie auch selbst ein Thema auswählen.

- Wie sehen gute Passwörter aus?
- Wie erkenne ich Phishing-E-Mails?
- Woran erkenne ich gefälschte Online-Shops?
- Bei welchen App-Berechtigungen sollte ich vorsichtig sein?
- Wieso sind unbekannte USB-Sticks oder USB-Ventilatoren gefährlich?
- Wie kann ich mich gegen Social Engineering schützen?

8 Zwei-Faktor-Authentifizierung

Oft wird bei Anmeldeverfahren zur zusätzlichen Sicherheit ein zweiter Faktor verlangt. Neben dem Passwort, muss dann beispielsweise der Login-Vorgang durch eine SMS-TAN bestätigt werden. Erklären Sie, warum ein zweiter Faktor die Sicherheit erhöht, und beschreiben Sie mögliche Vor- und Nachteile von verschiedenen Arten der Zwei-Faktor-Authentifizierung (z. B. App, SMS, Fingerabdruck) hinsichtlich Ihrer Sicherheit sowie des Datenschutzes.

Zum Weiterlesen

L8 Hacks und Datenlecks

Hack des deutschen Bundestages

Nicht nur gegen Unternehmen, auch gegen den deutschen Bundestag kam es bereits zu Hackerangriffen. Etwa im Dezember 2014 oder Januar 2015 hackten sich Ermittlungen zufolge Kriminelle im Auftrag Russlands in das IT-System des Parlaments. Zugang erlangten die Hacker vermutlich über Phishing-E-Mails, in denen sie sich als Mitarbeiter der Vereinten Nationen ausgegeben hatten. Hinter dem Link in dieser E-Mail war jedoch eine Seite verborgen, die unbemerkt Schadsoftware installierte. Andere Fachleute gehen hingegen davon aus, dass ein externer Webserver, dessen Dienst die Abgeordneten regelmäßig nutzen, manipuliert und auf die Auslieferung von Schadsoftware eingestellt wurde. Über diesen Trojaner konnten sich die Angreifenden schließlich im internen Netzwerk der Bundestags-IT ausbreiten.
Entdeckt wurde der Angriff erst im Mai 2015. Im Rahmen der Beseitigung der Schadsoftware musste schließlich das gesamte Netzwerk abgeschaltet werden. Schätzungen zufolge konnten die Kriminellen rund 16 GB an Daten (u. a. vertrauliche E-Mails) abgreifen. Auch wenn zunächst kein direkter Schaden entstanden ist, können vertrauliche Daten bei passender Gelegenheit eingesetzt werden, um politische Ziele durchzusetzen, indem bestimmte Personen damit erpresst werden.

Datenleck bei Facebook

Am 3. April 2021 entdeckte die IT-Sicherheitsfirma Hudson Rock, dass in einem Hacker-Forum eine sehr große Menge an Facebook-Nutzerdaten veröffentlicht worden waren. Insgesamt waren 533 Millionen Facebook-Nutzerinnen und -Nutzer betroffen, darunter über sechs Millionen aus Deutschland. Die Datensätze enthielten unterschiedliche Informationen wie beispielsweise den Nutzernamen, die Facebook-ID, den vollständigen Namen, die Telefonnummer, das Geburtsdatum oder andere Angaben wie den Beziehungsstatus und die E-Mail-Adresse. Angeblich enthielten die Daten auch die Telefonnummer von Facebook-Gründer Mark Zuckerberg.
Facebook versuchte den Vorfall kleinzureden und betonte, es handle sich um „alte" Datensätze, die bereits im Jahr 2019 gestohlen worden seien. Laut dem Konzern ist das System nicht gehackt worden, sondern die Daten wurden durch sogenanntes Scraping gesammelt, also durch automatisierte Abrufe von sehr vielen (öffentlich zugänglichen) Daten. Zudem würden die Datensätze keine Finanzinformationen, Gesundheitsinformationen oder Passwörter enthalten.
Klar ist jedoch, dass Kriminelle eine Sicherheitslücke ausnutzten, um an die Daten zu gelangen. Bereits zuvor gab es einige Meldungen zu Datendiebstählen und Sicherheitsproblemen bei Facebook: Im Jahr 2019 wurde beispielsweise bekannt, dass die Passwörter von Facebook unverschlüsselt gespeichert waren und so von Mitarbeitenden über Jahre hinweg lesbar waren. 2018 verschafften sich Hacker durch eine Sicherheitslücke Zugriff auf 50 Millionen Daten.
Die Vorfälle zeigen, dass große Konzerne keinesfalls sicher vor Angriffen sind. Im Gegenteil, bekannte Systeme mit vielen gespeicherten Daten ziehen sogar eher Kriminelle an als unbekannte Plattformen.

Daten von Jugendamt und Kfz-Zulassungsstelle bei eBay

Im Oktober 2019 kaufte ein eBay-Nutzer auf der Plattform eine Festplatte. Laut Artikelbeschreibung sollten diese in einwandfreiem Zustand sein. Nach Erhalt der Ware stellte der Käufer nicht nur fest, dass die Festplatte Gebrauchsspuren hatten. Er stellte auch fest, dass auf dem Laufwerk eine gesamte Windows-10-Installation mit vielen Daten gespeichert war. Er übergab das Gerät daher an die Redaktion des Magazins für Computertechnik (c't). Diese untersuchte die Hardware eingehend. Dabei fand sie zahlreiche Daten aus der Zulassungsstelle Coburg. Neben Dokumenten rund um die An- und Abmeldung von Fahrzeugen konnte die Redaktion auch eine Vielzahl von E-Mails entdecken. Diese enthielten Kommunikation mit Bürgerinnen und Bürgern. In den Anhängen der Mails waren Vollmachten, Versicherungsdaten, Handelsregisterauszüge, Bußgeldbenachrichtigungen, Zwangsversteigerungen und weitere sensible Daten gespeichert. In den E-Mails waren aber auch Passwörter für Systeme wie dem europäischen Fahrzeug- und Führerscheininformationssystem EUCARIS oder für das Zentralregister des Kraftfahrt-Bundesamtes REGINA zu finden. Auch Zugangsdaten zu Datenbanken wie der DEKRA-Gutachten-Datenbank waren gespeichert.
Zudem ergab die Recherche, dass eine Mitarbeiterin der Zulassungsstelle einen Teil ihrer Arbeitszeit für das Jugendamt Coburg aufwendete. Diese Arbeit wurde scheinbar am gleichen PC ausgeführt, so dass auch E-Mails zu Jugendamtsangelegenheiten wie Betreuungsverhältnisse, Heimunterbringungen und Unterhaltsfragen zu finden waren.
Von dem Vorfall in Kenntnis gesetzt, leitete der bayerische Datenschutzbeauftragte Ermittlungen ein und der Datenschutzbeauftragte des Landkreisamtes Coburg schaltete die Kriminalpolizei ein.
Ein IT-Dienstleister hatte die Festplatten im Landratsamt per Auftrag ausgetauscht. Nachdem der Vorfall des Datenfunds veröffentlicht worden war, erklärte das Amt, der Dienstleister hätte die Laufwerke nicht fachgemäß gelöscht. Der Dienstleister wiederum behauptete, das Landratsamt hätte in der Ausschreibung explizit auf die Löschung verzichtet.
Unabhängig davon, welche Version richtig ist, hat das Landratsamt fahrlässig und entgegen der DSGVO gehandelt, indem es personenbezogene und hochsensible Daten unverschlüsselt gespeichert hat.

L9 Das Bundesamt für Sicherheit in der Informationstechnik

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) mit Sitz in Bonn ist die Cyber-Sicherheitsbehörde der Bundesrepublik Deutschland. Die über 1300 Mitarbeitenden (Stand 2021) sind nicht nur verantwortlich für die Sicherheit der IT-Systeme der öffentlichen Hand, sondern arbeiten auch daran, den sicheren Einsatz von IT-Systemen in Wirtschaft und Gesellschaft voranzubringen. So gibt das BSI u. a. regelmäßig Mindeststandards und Handlungsempfehlungen zur IT-Sicherheit heraus, die sich teils an Privatpersonen, teils an Firmen richten. Auf der Webseite des BSI finden sich daher auch für Privatpersonen zahlreiche Tipps und Empfehlungen zu unterschiedlichsten Themen rund um die IT-Sicherheit. Zudem können verschiedene Newsletter abonniert werden, die z. B. aktuelle Sicherheitshinweise enthalten.
Mit dem CERT-Bund (Computer Emergency Response Team für Bundesbehörden) verfügt das BSI außerdem über ein Notfallteam, das bei IT-Sicherheitsvorfällen in Behörden hilft. Neben präventiven Aufgaben wie Veröffentlichung von Handlungsempfehlungen oder Hinweis auf Schwachstellen bietet es eine 24-Stunden-Rufbereitschaft.

L10 Glossar wichtiger Begriffe

Die folgende Übersicht erklärt einige Begriffe aus dem Kapitel und dient als Nachschlagewerk. Erweitern Sie die Sammlung nach Bedarf in Ihren Aufzeichnungen.

Advanced Persistent Threats: (kurz APT; deutsch: fortgeschrittene andauernde Bedrohung); gezielter Angriff auf ein Unternehmen oder auf eine staatliche Einrichtung über einen längeren Zeitraum, mit dem Ziel, möglichst lange unentdeckt im System zu bleiben, um Spionage zu betreiben oder anderweitigen Schaden anzurichten

Backup: (möglichst regelmäßige) Sicherung von Daten

Botnetz: viele, zu einem System zusammengeschlossene, Rechner, welche gemeinsam zu bestimmten Aktionen missbraucht werden, teilweise ohne dass die Besitzerinnen und Besitzer der Geräte dies mitbekommen

Denial-of-Service (**DoS**): gezielte Bombardierung eines Servers mit so vielen Anfragen, dass dieser im schlimmsten Fall ausfällt

Distributed-Denial-of-Service (**DDOS**)**:** spezielle Form eines DoS-Angriffes, bei dem (oft mit Botnetzen) viele Rechner zusammen ein Ziel angreifen

Drive-by-Exploits: Ausnutzung von Sicherheitslücken in Browsern oder Betriebssystemen, um beim Aufruf einer Webseite automatisch Schadsoftware herunterzuladen (daher auch mit Drive-by-Download bezeichnet)

Datenschutz-Grundverordnung (**DSGVO**): regelt unter anderem den Umgang mit personenbezogenen Daten in der europäischen Union

Full Disclosure: Veröffentlichung einer Sicherheitslücke unmittelbar nach dem Auffinden und ohne Rücksprache mit den verantwortlichen Stellen

Hackback: (staatlicher) Hackerangriff als Reaktion auf einen Hack eines anderen Staates

Malware: Schadsoftware, die unerwünschte oder gar schädliche Funktionen ausführt

Man-in-the-Middle-Attacke: Angriff auf Rechnernetze, bei dem der Angreifende physisch oder logisch zwischen den Kommunikationspartnern „sitzt" und so den gesamten Datenverkehr mitlesen oder manipulieren kann

Passwortmanager: Software zum Verwalten und Erstellen von sicheren Passwörtern

Pentest: Kurzform von Penetrationstest: intensive Sicherheitstests eines Systems, um Sicherheitslücken aufzudecken

Ransomware: Schadsoftware, welche die Daten des angegriffenen Systems verschlüsselt, um eine Lösegeldforderung zum Entschlüsseln zu stellen

Responsible Disclosure: Veröffentlichung einer Sicherheitslücke erst nach intensiver Rücksprache mit den verantwortlichen Stellen, damit die Lücke vor der Veröffentlichung geschlossen werden kann

Sandbox: abgeschirmte Umgebung, in der Software getestet werden kann, ohne dass es Auswirkungen auf das System gibt

Scraping: automatisiertes Sammeln von vielen Daten

Security Awareness: Bewusstsein für IT-Sicherheit

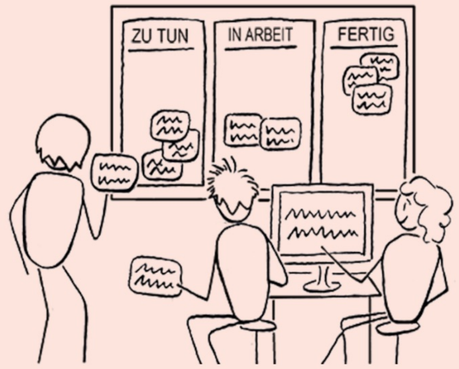
Security-Token: Hardware-Komponente zur Identifizierung und Authentifizierung von Nutzern

Social Engineering: gezielte Manipulation von Menschen, um beispielsweise an vertrauliche Informationen zu gelangen, mit welchen ein IT-System angegriffen werden kann

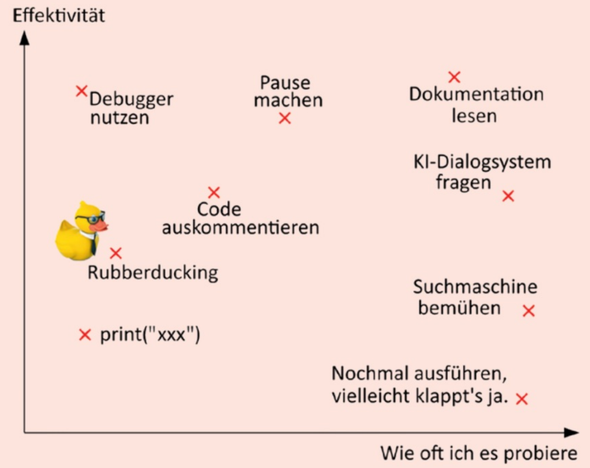
6 Softwareentwicklung

In diesem Kapitel können Sie nachlesen, wie man ...

... Projekte plant und durchführt.



... Debuggingstrategien passend auswählt und einsetzt.



... Programmbibliotheken gewinnbringend einsetzt.

6.1 Im Team arbeiten: Agile Methoden

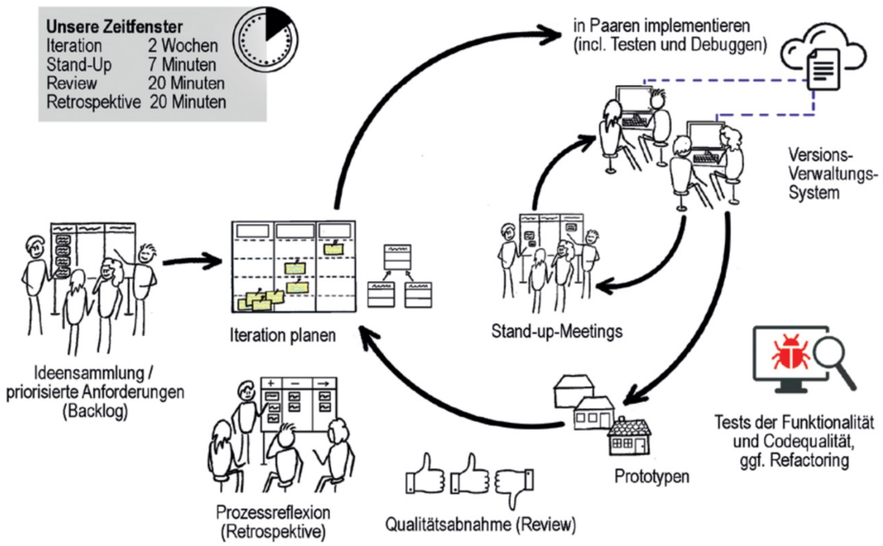
Planung, Kommunikation und Transparenz als Schlüssel zum Erfolg

→lat. iterare: wiederholen; in der Softwareentwicklung werden Iterationen auch als Sprint bezeichnet.

Softwareprojekte werden meist im Team bearbeitet, weil die Fertigstellung sonst zu lange dauern würde und unterschiedliche Erfahrungen und Sichtweisen Einzelner dabei helfen, komplexe Aufgabenstellungen zu lösen. Eine typische Organisationsform besteht aus sich wiederholenden festen Zeitfenstern (→**Iterationen**), in denen das Team einen →**Prototyp** jeweils um weitere Funktionalitäten ergänzt. Jede Iteration umfasst

- eine Planungsphase, in der das Ziel des aktuellen Durchlaufs festgelegt wird,
- eine Arbeitsphase, in der auch kurze Besprechungen und Testphasen enthalten sind, sowie
- eine Reflexion der Qualität und des Arbeitsprozesses.

→Prototyp: funktionsfähiges Produkt, das fehlerfrei ist, jedoch nicht den vollen Funktionsumfang hat



→engl. agile: wendig, beweglich (weil zu Beginn jeder Iteration die Schwerpunkte neu gesetzt werden können)

→lat. incrementum: Wachstum, Zunahme

Während einer Iteration sollte ihr Ziel nicht mehr geändert werden, damit das Team fokussiert arbeiten kann. Vorteile dieser Vorgehensweise sind früh sicht- und testbare Zwischenergebnisse. Verankert sind auch klare Kommunikationsstrukturen und eine Visualisierung der Planung und des Arbeitsstands an einem Project Board.
Wie in der Abbildung oben dargestellt, wiederholen sich bei der →agilen Vorgehensweise Phasen und mit jeder Iteration hat das Produkt mehr Funktionalitäten. Deswegen spricht man von einer iterativ →inkrementellen Softwareentwicklung. Im Folgenden sind die einzelnen Phasen näher erläutert.

Grobplanung – →Backlog

- Notieren Sie jede Anforderung an das Produkt als sogenannte **User-Story** mit einem Titel und einer knappen Beschreibung aus Nutzersicht (siehe Beispiele zu einem Abischnittrechner rechts). Achten Sie auf kleine Aufgabenpakete!
- Entscheiden Sie über die Wichtigkeit der User-Story auf einer Skala von 1 bis 50 (**Priorisierung**): Je kleiner die Zahl, desto wichtiger und desto früher wird die Anforderung umgesetzt. Lücken in den Prioritäten verdeutlichen die unterschiedliche Wichtigkeit.
- Häufig muss man sich in Projekten auch in neue Bereiche einarbeiten, Wissen aufbauen oder neue Techniken kennenlernen. Für Ihr Softwareprojekt können dies beispielsweise graphische Benutzeroberflächen, Softwareergonomie, persistente Datenspeicherung sowie Entwurfsmuster sein. Da das Lernen auch in der Zeitplanung berücksichtigt werden muss, sollte dies auch als „Aufgabenpaket" verschriftlicht werden.
- Ein **Project Board** mit den Spalten „ZU TUN", „IN ARBEIT" und „FERTIG" gibt einen Überblick über die Ziele und den aktuellen Bearbeitungsstand. Bereiten Sie das Project Board vor, indem Sie die Aufgabenpakete in der Reihenfolge der Priorisierung (wichtigste oben) anheften.

Hinweis: Am Ende einer Iteration dürfen Aufgaben im Backlog ergänzt bzw. die Priorisierung geändert werden. Während einer Iteration ist dies üblicherweise nicht erlaubt!

Speicherung der Fächerbelegung

Für Nutzerinnen und Nutzer ist es möglich, die Belegung bestehend aus Fach, Belegung in Jgst. 12 und/oder 13, Abiturfach ja/nein zu speichern. 2

Eingabe der Noten

Zu allen belegten Fächern können eine Klausurnote und mehrere kleine Leistungsnachweise eingegeben bzw. geändert werden. 12

Pflichtbelegungen

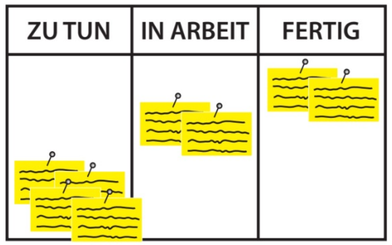
Pflichtbelegungen (Mathematik, Deutsch, W-Seminar, ...) sind bereits gespeichert, damit Nutzende weniger eingeben müssen. 20

Speichern in einer Datenbank

Recherchieren und in einem Test-Projekt umsetzen: Mit Java/Python/Swift in einer Datenbank Daten einfügen, abrufen und ändern. 5

Erstellen eines Repositorys ...

... und Verteilen des Zugangs an alle Teammitglieder. 1



→Backlog: priorisierte Liste von zu erledigenden Aufgaben in einem Projekt; enthält alle vom Kunden gewünschten Funktionalitäten eines Produkts (Produkt-Backlog)

→6.4
→6.5
→6.6

Organisatorische Festlegungen, insbesondere Festlegung von Zeitfenstern
(gibt ggf. die Lehrkraft vor)

- Iterationsdauer: typischerweise 2 Wochen
- Besprechungen: typischerweise Iterationsplanung, Stand-up-Meetings (Kurzbesprechungen von maximal 7 Minuten im Stehen), Reflexionsmeeting (**Retrospektive**)
- Vorgaben zur technischen Umsetzung (z. B. Entwicklungsumgebung), zum Endprodukt (dass z. B. eine persistente Datenspeicherung umgesetzt sein muss) und zur Dokumentation
- Qualitätsabnahme (**Review**): Vorstellen der Zwischenergebnisse, typischerweise alle 2–3 Iterationen
- Zeitrahmen, nach dem die Rollen beim Pair-Programming (s. nächste Seite) gewechselt werden

Implementierung eines fehlerfreien Prototyps

- Wählen Sie entsprechend der Priorisierung User-Storys für die Iteration aus.
- User-Storys sind aus der Nutzersicht formuliert. Wechseln Sie nun die Perspektive und notieren Sie in sogenannten **Tasks** die Aufgabenpakete aus Entwicklersicht. Oft ergeben sich zu einer User-Story mehrere Tasks. Entscheiden Sie vorab im Team, ob diese Detailplanung durch das gesamte Team oder durch das verantwortliche Paar erfolgt.

Speicherung der Fächerbelegung

Für Nutzerinnen und Nutzer ist es möglich, die Belegung bestehend aus Fach, Belegung in Jgst. 12 und/oder 13, Abiturfach ja/nein zu speichern. 2

Klasse BELEGUNG erstellen …

… zum Speichern aller Fächer in einem Feld.
mit Methoden:
BelegungÄndern (fach, belegung12, belegung13, abi)
BelegungAusgeben()

Klasse FACH erstellen …

… mit den passenden Attributen, Getter- und Setter-Methoden.

Zwei Tasks zur User-Story „Speicherung der Fächerbelegung"
Beachten Sie: Informatische Fachbegriffe sind in einer User-Story unerwünscht (Nutzersicht), in Tasks sind sie erforderlich (Entwicklersicht).

- Jede Unterrichtsstunde startet mit einem Stand-up-Meeting in den einzelnen Gruppen, bei dem reihum jedes Paar folgende Fragen beantwortet:
  - Welche Tasks wurden seit dem letzten Treffen erledigt?
  - Welcher Task wird als nächstes bearbeitet?
  - Falls Probleme aufgetreten sind, welche Probleme waren es?

Wichtig ist es, die Besprechung kurz zu halten. Es geht nur um einen Überblick, keine Diskussion. Bei Problemen kann im Stand-up ein Ansprechpartner gefunden werden, der im Nachgang Hilfestellung leistet.

- Die Umsetzung erfolgt zu zweit (**Pair Programming**). Jedes Paar nimmt einen Task, ergänzt auf der Karte die eigenen Namen als Verantwortliche und hängt sie in die Spalte IN ARBEIT auf dem Project Board. Entsprechend den Rollen **Driver**/**Navigator** wird implementiert. Die Rollen werden regelmäßig gewechselt!
- Vor dem Umhängen des Tasks in die Spalte FERTIG wird getestet!
- Gibt es eine Schwierigkeit, die mehrere Paare betrifft (z. B. Methodennamen in unterschiedlichen Klassen passen nicht zusammen), darf spontan ein Stand-up-Meeting einberufen werden.

Der Driver …
- verwendet Tastatur und Maus und verfasst den Quelltext.
- „denkt laut", d. h. teilt seine Absichten dem Navigator mit.

Der Navigator …
- stellt regelmäßig Fragen.
- achtet auf Lesbarkeit des Quelltextes, z. B. durch aussagekräftige Namen für Methoden und Variablen.
- achtet auf eine gute Strukturierung des Quelltextes.
- ist verantwortlich für die Fokussierung auf die aktuelle User-Story.

Qualitätssicherung

→ 6.8

- Sind alle Tasks einer User-Story bearbeitet, wird die Dokumentation und Lesbarkeit des Quellcodes überprüft und falls nötig verbessert. Weiterhin wird kontrolliert, ob alle Anforderungen erfüllt sind.
- Ist eine neue Funktionalität fertig programmiert, so wird diese in den Gesamtkontext eingebunden und der Quelltext in das Server-Repository hochgeladen. Tests nicht vergessen! Falls alles funktioniert, wird die User-Story in die Spalte FERTIG umgehängt (und Sie dürfen jubeln – Erfolge muss man feiern).

Reflexion

- Je nach Vereinbarung wird die Funktionsweise, Strukturierung und ein zentraler Quelltextausschnitt des Prototyps anderen vorgestellt (Review). Aufgabe der Zuhörerschaft ist es, eine kritisch konstruktive Rückmeldung zu geben.
- Am Ende jeder Iteration wird in der Retrospektive der Arbeitsprozess reflektiert: Was hat gut funktioniert? Wo gab es Schwierigkeiten? Was können wir lernen und in der nächsten Iteration verbessern? Bei Bedarf gibt Ihnen die Lehrkraft auch einen Fokus für die Retrospektive. Seien Sie offen und ehrlich zueinander und halten Sie Ihr Ziel für die nächste Iteration schriftlich fest.

Ergebnis Retrospektive Iteration 2
☺ gute Absprache bei den Schnittstellen
☹ Arbeitsfortschritt zu gering

Ziele für die nächste Iteration
· Alle sind mit Gongschlag eingeloggt.
· Zeitwächter beim Stand-up

Agiles →Manifest – Agile Werte

→lat. manifestus: handgreiflich gemacht, offenbart, also öffentliche Erklärung von Zielen und Absichten

Im Jahr 2001 haben Praktiker aus dem Bereich der Softwareentwicklung Leitsätze und Prinzipien formuliert, die Vorgehensweisen und Werte bei der Softwareerstellung beschreiben. Dies verdeutlicht, dass agile Methoden nicht nur unterstützende Techniken bei der Projektorganisation und -durchführung sind, sondern auch eine Denk- und Verhaltensweise, wie man im Team miteinander arbeitet und kommuniziert.

Leitsätze des Agilen Manifests

- Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Funktionierende Software ist wichtiger als umfassende Dokumentation.
- Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlung.
- Reagieren auf Veränderung ist wichtiger als das Befolgen eines Plans.

Agile Werte

- **Selbstorganisation**
  Organisiert zusammen im Team euren Arbeitsprozess so, dass ihr bestmöglich eure Ziele erreicht. Besprechungen helfen dabei. Hol dir aktiv Aufgaben vom Project Board, die du gut bewältigen kannst.
- **Einfachheit**
  Wähle die einfachste Lösung, um den größtmöglichen Projektfortschritt (Erfüllung von User-Storys) zu erhalten: Keep it small and simple (KISS-Prinzip)
- **Mut**
  Sei mutig, Fehler zu machen, denn jeder Fehler bietet die Chance, etwas zu lernen. Sei mutig, Fragen zu stellen, Verantwortung zu übernehmen, Entscheidungen zu treffen und auch ehrlich über Misserfolge und Hindernisse zu sprechen.
- **→Commitment**
  Übernimm mit deinem Team Verantwortung sowohl für die Produktentwicklung als auch für den Lernfortschritt. Dazu gehört Engagement, das Gestalten von Freiheiten und das gemeinsame Festlegen von Zielen mit einem „Ja, wir wollen das!".
- **Zielstrebigkeit und Fokussierung**
  Fokussiere deine Aufmerksamkeit, deine Bemühungen und Fähigkeiten darauf, um in dem vorgegebenen Zeitfenster das zugesagte Ziel zu erreichen.

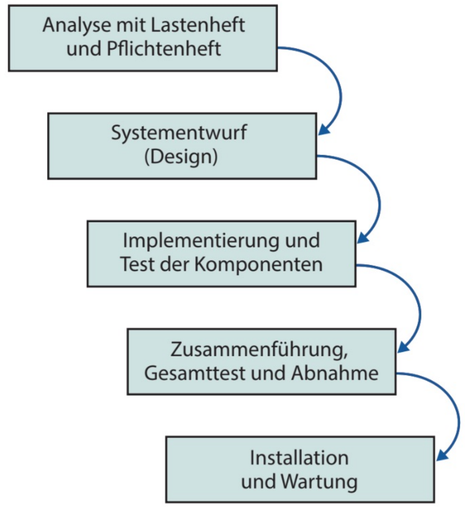
→engl. commitment: Zusage, Versprechen, Verpflichtungserklärung

6.2 Projekte organisieren: Das Wasserfallmodell

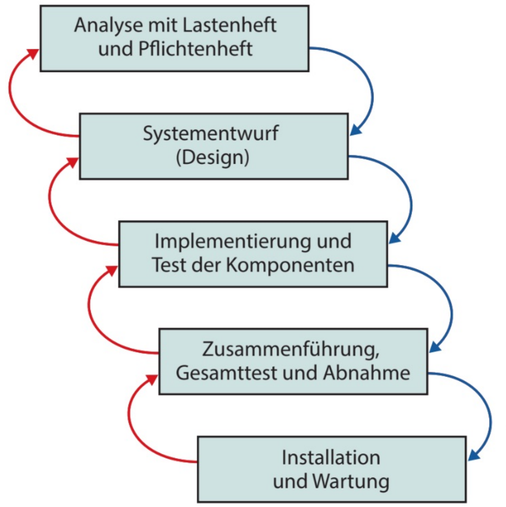
Agile Methoden sind eine beliebte, aber nicht die einzige Möglichkeit, die Arbeit in Projekten zu organisieren. Eine andere Möglichkeit bietet das sogenannte Wasserfallmodell, das in der Bau- und Produktionsplanung verwendete Prozesse auf die Softwareentwicklung überträgt. Beim **Wasserfallmodell** wird die Entwicklung eines großen Projekts in mehrere Schritte aufgetrennt, die Entwicklungsphasen genannt werden. Jede Phase des Projekts muss abgeschlossen sein, ehe die nächste Phase beginnen kann. Der Name des Modells erklärt sich aus der typischen grafischen Darstellung der Phasen, die einem Wasserfall ähnelt.

In untenstehender Abbildung sind die typischen Phasen des Wasserfallmodells angegeben:

- **Analyse** (Anforderungen): Der Auftraggeber beschreibt die gewünschten Leistungen im **Lastenheft**.
  Der Auftragnehmer legt im **Pflichtenheft** fest, auf welche Art er diese Leistungen erbringen will. Damit ist der genaue Auftrag festgelegt.
- **Systementwurf** (Design): Die Struktur des geplanten Softwarepakets wird erstellt. Die einzelnen Teile werden in Leistungsumfang und Schnittstellen genau beschrieben (spezifiziert).
- **Implementierung**: Die einzelnen Teile werden in der Regel in parallel arbeitenden Gruppen erstellt; jedes Teil wird vollständig auf seinen geforderten Leistungsumfang hin getestet.
- **Test, Bewertung und Abnahme**: Die einzelnen Teile werden zusammengeführt; das Gesamtprodukt wird auf Fehlerfreiheit getestet. Dabei wird auch festgestellt, ob es die vereinbarten Pflichten in vollem Umfang erfüllt.
- **Installation und Wartung**: Das Produkt wird beim Auftraggeber installiert und genutzt. Oft kommt es zur Weiterentwicklung des Produkts, hervorgerufen durch aufgetretene Mängel, neue Anforderungen oder Änderung in den Rahmenbedingungen wie neue Hard- oder Software, mit denen das Produkt genutzt wird.



In der Praxis hat sich allerdings gezeigt, dass das einfache Wasserfallmodell Schwächen hat. Trotz sorgfältiger Vorgehensweise stellt sich manchmal erst beim Entwurf oder gar bei der Implementierung heraus, dass Anforderungen des Pflichtenhefts so nicht umsetzbar sind. Oft zeigt sich auch erst in der Implementationsphase, dass Schnittstellen nicht den tatsächlich benötigten Informationsaustausch zwischen den Teilen des Projekts ermöglichen. Da es viel zu teuer (und auch unnötig) ist, alles bisher Entwickelte wegzuwerfen und in der Phase neu zu beginnen, in der der Fehler aufgetreten ist, wurden Vorgehensweisen entwickelt, die ein gezieltes Zurückgehen unter Beibehaltung von möglichst viel bereits geleisteter Arbeit ermöglichen. Dieses ergänzte Vorgehensmodell (siehe nebenstehende Abbildung) heißt **Wasserfallmodell mit Rückkopplung** oder erweitertes Wasserfallmodell.



Trotzdem kostet ein Zurückgehen immer Zeit (und damit Geld); eine sorgfältige Planung minimiert die Häufigkeit solcher unerwünschten Maßnahmen.

Wasserfallmodell versus agile Methoden

Das Wasserfallmodell eignet sich besonders für Projekte mit klaren Anforderungen und kurzer Laufzeit. Hier ist das strukturierte Vorgehen mit einer sehr präzisen Analyse und Definition der Anforderungen zielführend für einen Projekterfolg. So kann die Umsetzung sehr gut geplant und der Aufwand abgeschätzt werden. Dagegen ist es in umfangreichen Projekten mit komplexen Einsatzbereichen nachteilig, Anforderungen früh festzulegen und die Umsetzung detailliert zu planen, weil eine Umplanung in späten Phasen – z. B. wegen technischer Weiterentwicklungen, unvorhergesehener Probleme, veränderter Kundenerwartungen, fehlerhafter Anforderungsfestlegung – teuer ist.

Diesen Schwierigkeiten begegnet das agile Vorgehen durch das mehrfach iterative Durchlaufen der Entwicklungsphasen. In jeder Iteration …

- … kann der Kunde in Absprache mit dem Entwicklungsteam die User-Storys neu priorisieren bzw. auch neue festlegen und in die Projektplanung aufnehmen.
- … kann der Kunde am Ende den aktuellen Prototyp mit seinen Erwartungen vergleichen und dem Entwicklungsteam ein Feedback geben.

Diesen Vorteilen steht gegenüber, dass gegebenenfalls durch die fehlende Detailplanung in der Anfangsphase im Vergleich zum Wasserfallmodell mehr Schnittstellenanpassungen und Refactoring im Projektverlauf durchgeführt werden müssen.

→ 6.8

6.3 Das Rad nicht neu erfinden: Bibliotheken nutzen

Welche Module über einen import-Befehl eingebunden werden müssen und welche nicht, hängt stark von der Programmiersprache ab.

Bei vielen Standardaufgaben wie dem Speichern von gleichartigen Daten in einer Liste, dem Lesen und Schreiben von Daten in eine Datei oder dem Verarbeiten von Zeichenketten muss man nicht bei „Null" beginnen, sondern man kann dazu bereits programmierte Klassen einfach nutzen. Häufig bezeichnet man ein solches thematisch ausgerichtetes Modul aus ein oder mehreren Klassen als (Programm-)Bibliothek.

Zwei notwendige Voraussetzung gibt es für die Nutzung solcher Klassen: Einerseits muss die Bibliothek im eigenen Programm eingebunden werden, d. h. die Klassen der Bibliothek müssen im eigenen Programm bekannt gemacht werden. Dies ist beispielsweise durch einen import-Befehl möglich. Andererseits muss man die Methoden als Schnittstelle zwischen der Bibliothek und dem eigenen Programm kennen. Zu jeder Anwendungsprogrammierschnittstelle, kurz →API genannt, gibt es eine Dokumentation. Typischerweise enthält diese zu jeder Methode den Bezeichner, die Parameterliste, gegebenenfalls den Datentyp der Rückgabe sowie eine kurze Erklärung:

→API application programming interface

Java
```
import java.util.ArrayList;
```

Python
```
import pygame
```

Swift
```
import Foundation
```

Eigentlich kenne ich das schon: Bei Objekten der Klasse FELD bzw. LISTE nutze ich genau diese Methoden schon länger ...

Klasse LISTE für Elemente vom Datentyp E	
Hinzufügen(index: GANZZAHL, element: E)	Fügt das Element an der durch den Index festgelegten Stelle in die Liste ein.
IstLeer() -> WAHRHEITSWERT	Gibt WAHR zurück, falls die Liste keine Elemente enthält, sonst FALSCH.
ElementGeben(index: GANZZAHL) -> E	Gibt das Element mit dem angegebenen Index zurück.
Entfernen(index: GANZZAHL) -> E	Entfernt das Element mit dem angegebenen Index und gibt es zurück.
ElementanzahlGeben() -> GANZZAHL	Gibt die Anzahl der Elemente in der Liste zurück.
...	...

Verwendet man ein Framework z. B. zur Spielentwicklung oder zum Erstellen einer Simulation, muss man auch hier eine passende API kennen, damit man – wie im folgenden Beispiel eines Taktgebers – die Methoden überhaupt nutzen kann:

Klasse EREIGNISBEHANDLUNG	
Anhalten()	Hält den Taktgeber an.
Starten()	Startet den Taktgeber.
TaktdauerSetzen(dauer: GANZZAHL)	Setzt die Taktdauer in Millisekunden und stellt damit die Ablaufgeschwindigkeit ein.
TaktImpulsAusführen()	Führt den Taktimpuls aus. Dazu werden alle Anweisungen im Rumpf dieser Methode ausgeführt.

Bei der Arbeit mit Bibliotheksklassen kann es sehr hilfreich sein, KI-Dialogassistenzsysteme als Werkzeug zu benutzen, die Codevorschläge liefern. Für die Anpassung an die selbst geschriebenen Klassen benötigt man dann aber zumindest ein Grundverständnis für die Struktur und Wirkungsweise.

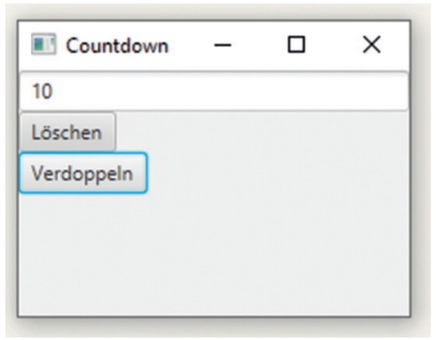
Kannst du mir eine Java →GUI schreiben mit zwei Buttons und einem Textfeld? Auf dem ersten Button soll löschen stehen, auf dem zweiten verdoppeln. Und dann brauchen wir noch einen Timer, der von 10 sekündlich bis 0 abwärts zählt.

→engl. grafical user interface: grafische Benutzeroberfläche

Natürlich kann ich dir eine Java-GUI schreiben, die zwei Buttons, ein Textfeld und einen Timer enthält. Hier ist ein Beispielprogramm:

```
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Duration;

public class CountdownGUI extends Application {

    private int count = 10;
    private TextField textField;

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("Countdown");

        textField = new TextField();
        textField.setText(Integer.toString(count));
        …
```

Das muss man noch verbessern, sieht aber für's Erste schon ganz gut aus! Und für die Ereignisse hat das System auch einen Vorschlag gemacht!

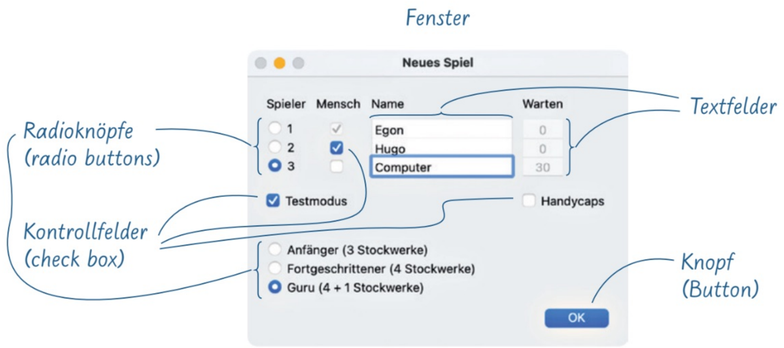


6.4 Grafische Benutzeroberflächen

Grundelemente grafischer Benutzeroberflächen

→GUI: Grafical User Interface

Das wichtigste Hilfsmittel zur intuitiven Bedienbarkeit von Computerprogrammen stellen die seit den 1980er-Jahren entwickelten grafischen Benutzeroberflächen (→ GUI) dar.
Das genaue Aussehen grafischer Benutzeroberflächen differiert zwischen den verschiedenen Betriebssystemen, deren Versionen und manchmal auch zwischen den verwendeten Programmiersprachen. Die meisten Oberflächenelemente sind aber immer leicht zu erkennen, überall vorhanden und in ihrer Funktion gleich.



Auf Desktop- und Laptopoberflächen werden die wichtigsten Anweisungen in der Menüzeile (menu bar) zusammengefasst; Smartphones und Tablets haben typischerweise eine andere Benutzerführung.

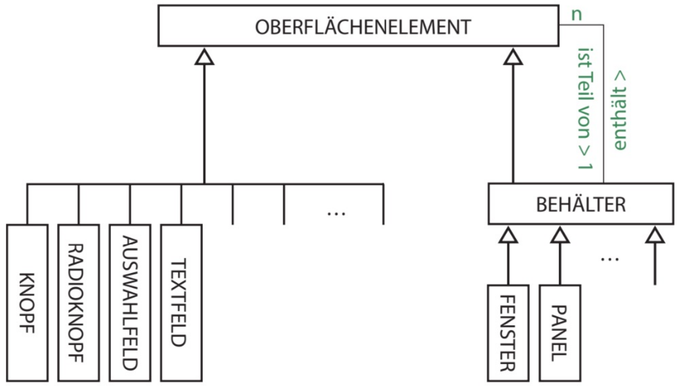
Weitere typische Oberflächenelemente sind Aufklappfelder (drop-down list, pop-up menus), Schieberegler (slider), Reiter (tabs), Rollbalken (scroll bars) und Werkzeugleisten (tool bars).

Jede Menge Klassen

Bei Verwendung objektorientierter Sprachen werden die Elemente einer grafischen Benutzeroberfläche in großen Klassenbibliotheken zusammengefasst. Die Klassenbibliotheken grafischer Oberflächen nutzen das Mittel der Vererbung intensiv, da viele Attribute und Methoden mehreren oder allen Elementen gemeinsam sind. An der Spitze der Klassenhierarchie steht eine Klasse, die alle grundlegenden Attribute für Position, Größe, Sichtbarkeit usw. und die zu deren Verwaltung benötigten Methoden zur Verfügung stellt (in der folgenden Abbildung OBERFLÄCHENELEMENT). Danach spaltet sich der Baum in zwei Äste auf. Diejenigen Elemente, die selbst wieder Unterelemente verwalten können (Fenster, Dialoge, Panele etc.), bilden den

einen Ast. An seiner Spitze steht eine Klasse (in der Abbildung BEHÄLTER), die als Zusatz die Verwaltung der untergeordneten Elemente als Enthält-Beziehung bereitstellt. Alle anderen, einfachen Elemente bilden den zweiten Ast. Oft sind diese Hierarchien noch weiter untergliedert, hier unterscheiden sich dann aber die Implementierungen der verschiedenen GUI-Bibliotheken. Dieser Klassenbaum entspricht dem Entwurfsmuster Kompositum, die grün dargestellte Beziehung ist die für das Muster Kompositium charakteristische, rekursive Enthält-Beziehung.

→ 6.6



Ereignisse und ihre Verwendung

Bei grafischen Oberflächen haben die Nutzenden oft mehrere Möglichkeiten, eine Botschaft zu senden. Diese Botschaften werden als Ereignisse bezeichnet, die von den Nutzenden ausgelöst werden. Die Nutzenden können mit der Maus einen Menüpunkt anwählen, eine Schaltfläche drücken, ein Optionsfeld wählen usw. Sie können auch über die Tastatur einen Text in einen Eingabebereich eingeben. Das Programm muss bereit sein, auf alle diese Tätigkeiten zu reagieren. Alle diese Botschaften werden aber von nur vier grundlegenden Eingabeereignissen ausgelöst: Taste gedrückt, Taste losgelassen, Mausknopf gedrückt und Mausknopf losgelassen.
Ergänzt werden diese Ereignisse mit Zusätzen wie Tastennummer und gedrückte Sondertasten bei den Tastenereignissen bzw. Mausposition bei den Mausereignissen. Allerdings macht es große Mühe, z. B. aus Tastennummer, Tastaturbeschreibung, Sondertasten und Landessprache zu ermitteln, welcher Buchstabe nun wirklich gemeint war. Daher bieten die meisten Oberflächenbibliotheken eine zweite, abstraktere Ereignisschicht an, auf der man Ereignisse wie „Buchstabe getippt" oder „Menüpunkt aufgerufen" oder „Knopf geklickt" erhält.
Um über ein eintretendes Ereignis informiert zu werden, muss bei dem gewählten Oberflächenobjekt (z. B. einem Knopf) ein Ereignisbehandler (Event Handler) als Beobachter angemeldet werden. Ereignisbehandler sind Objekte, die bestimmte Methoden (z. B. zur Reaktion auf Mausklick) mit vorgegebenem Methodenkopf bereitstellen müssen. Diese Methoden werden als Reaktion auf ein Ereignis aufgerufen.

→ 6.6

Oberflächeninhalte zeichnen

Jedes Oberflächenelement besitzt eine Methode *Zeichnen*, die aufgerufen wird, wenn sich das Element auf dem Bildschirm darstellen soll. Diese Methode hat in der Regel als Parameter ein Objekt einer Klasse GRAFIK, welches grundlegende Zeichenmethoden wie Linie zeichnen, Rechteck zeichnen, Kreis zeichnen, Text zeichnen usw. sowie das Einstellen von Farben, Mustern und Ähnlichem zur Verfügung stellt.

Tipp: Lassen Sie sich eine Oberfläche von einem geeigneten KI-Dialogsystem erstellen.

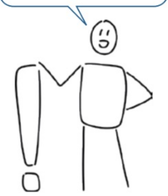
6.5 Daten persistent speichern: Datenbank oder Datei

Typischer Zugriff auf Datenbanken

Um aus objektorientierten Programmen heraus auf SQL-Datenbanken zuzugreifen, stehen in objektorientierten Sprachen drei Klassen nach dem folgenden Grundschema zur Verfügung.

DATENBANKVERBINDUNG	ANWEISUNG	ERGEBNISTABELLE
VerbindungÖffnen(name) AnweisungErzeugen(sql) VerbindungSchliessen()	AnweisungAusführen() AbfrageAusführen() AnweisungSchliessen()	AufNächstenDatensatzPositionieren() FeldwertLesen(feldname)

Oft müssen auch noch Server und/ oder Pfad und gegebenenfalls ein Passwort angegeben werden.



Mit `VerbindungÖffnen("mitglieder")` öffnet man eine Datenbankverbindung zur Datenbank mitglieder.

Durch den Code in der folgenden Zeile erzeugt man eine Abfrage auf der Tabelle Benutzer:

```
ANWEISUNG a = AnweisungErzeugen("SELECT id, nachname FROM Benutzer")
```

Diese Anweisung kann nun – auch wiederholt – ausgeführt werden. Da es sich um eine Abfrageanweisung handelt, erhält man eine Referenz auf ein Objekt vom Typ ERGEBNISTABELLE zurück:

```
ERGEBNISTABELLE e = a.AbfrageAusführen()
```

Danach beginnt man, diese Tabelle auszulesen.

id	nachname
4711	Kopf
815	Ne

Beim ersten Aufruf von e.AufNächstenDatensatzPositionieren() wird die erste Zeile der Tabelle zum aktuellen Datensatz. e.FeldwertLesen("nachname") gibt nun „Kopf" zurück, e.FeldwertLesen("id") liefert 4711.

Der zweite Aufruf von e.AufNächstenDatensatzPositionieren() positioniert auf die zweite Zeile der Tabelle.

Bei jedem Aufruf von e.AufNächstenDatensatzPositionieren() wird ein Wahrheitswert zurückgegeben. Ist dieser WAHR, so ist noch ein Datensatz verfügbar, ansonsten ist man nach dem letzten Datensatz angelangt.

Um die Ergebnistabelle komplett auszugeben, benötigt man den folgenden Code:

Aufruf liefert WAHR, solange das Ende der Tabelle nicht erreicht wird. So wird die Tabelle komplett durchlaufen und dann die Wiederholung beendet.

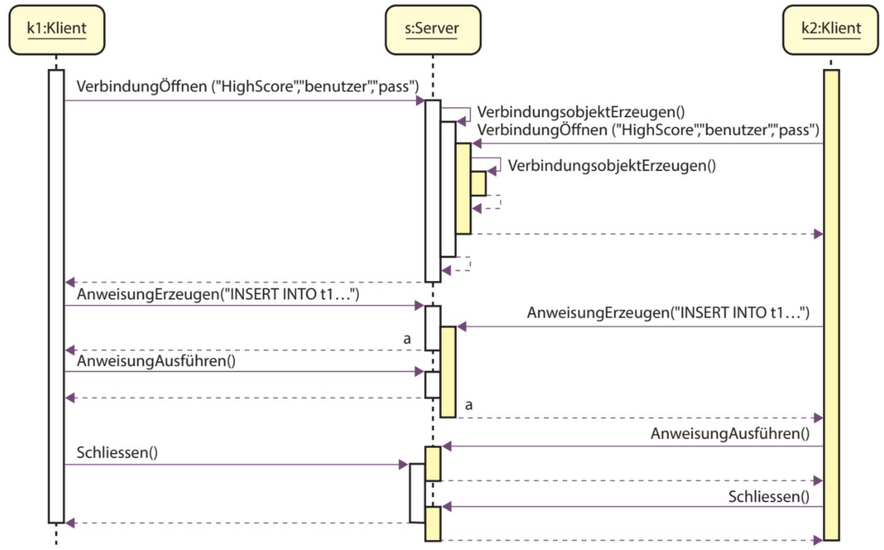
```
wiederhole solange e.AufNächstenDatensatzPositionieren()
   AusgebenZeile(e.FeldwertLesen("id") + e.FeldwertLesen("nachname"))
endewiederhole
```


Mehrbenutzerdatenbanken

Datenbanken können einem Programm fest zugeordnet sein und erfüllen dort ihre Aufgabe. In diesem Fall ist das Programm der alleinige Nutzer und es bedarf daher keiner weiteren Zugangskontrolle.

Die meisten Datenbanken sind aber für die Verwendung durch viele Personen gedacht. Hier wird natürlich eine Zugangskontrolle benötigt, die genau festlegt, welcher Benutzer welche Daten sehen bzw. verändern darf. Sie werden meist auf eigenen Servern verwaltet, der Zugriff erfolgt über das Internet. Für den Umgang mit der Datenbank ändert sich dadurch prinzipiell nichts. Statt eines einfachen Namens für das Öffnen der Datenbankverbindung werden hier zusätzlich zum Namen die Server-URL sowie Benutzername und Passwort angegeben, z. B.: „//server/datenbank?user=name&password=1234".

Intern muss aber beim Ausführen von Anweisungen viel mehr getan werden. Die Rechte der Benutzer müssen überwacht werden. Auch müssen alle Änderungen vollständig abgeschlossen sein, ehe die nächste Änderung, die sich auf die gleichen Daten bezieht (z. B. Einfügen in die gleiche Tabelle), ausgeführt werden kann.



Abfrageergebnisse werden nicht direkt an den Aufrufer der Abfrage geschickt, sondern auf dem Server gehalten, da die Ergebnistabellen sehr groß sein können.

Diese Ergebnistabellen werden erst freigegeben, wenn der Benutzer die zugehörige Anweisung schließt. Daher sollten alle Anweisungen auch beendet werden, wenn man sie nicht mehr benötigt.

Überblick über die SQL-Anweisungen
Für sämtliche Aufgaben bei der Erstellung und Nutzung von Datenbanken gibt es entsprechende SQL-Befehle. Man unterscheidet im Wesentlichen drei Gruppen.
Datenbeschreibungsanweisungen dienen zum Erstellen, Ändern und Löschen von Tabellen und deren Eigenschaften (Schlüssel, Spaltentypen etc.). Diese Anweisungen werden am einfachsten über ein grafisches Werkzeug ausgeführt; die damit erstellte Datenbank wird dann verwendet.
Die **Abfrageanweisung** (SELECT) wählt Werte aus einer oder mehreren Tabellen aus, bearbeitet sie gegebenenfalls und legt das Ergebnis in einer Ergebnistabelle ab.
Datenmanipulationsanweisungen dienen dem Einfügen, Ändern und Löschen von Datensätzen. Diese Aktionen können zwar auch mit einem graphischen Werkzeug durchgeführt werden; soll ein selbst entwickeltes Programm die Inhalte einer Datenbank aber automatisch modifizieren können, so müssen hierfür die entsprechenden SQL-Befehle verwendet werden, welche im Folgenden jeweils kurz erklärt werden.

Datensätze in eine Tabelle einfügen mit INSERT
Soll ein neuer Datensatz in eine bestehende Tabelle eingefügt werden, so ist dies mittels eines INSERT-Befehls möglich. Um beispielsweise einen Datensatz für Hund Bello in die Tabelle Haustier (name:ZEICHENKETTE, art:ZEICHENKETTE, geschlecht:ZEICHEN) einzufügen lautet der Befehl:

```
INSERT INTO Haustier (name, art)
VALUES ("Bello", "Hund")
```

Hier werden alle Spalten aufgezählt, in die ein Wert eingefügt werden soll.

vorher:

name	art	geschlecht
Fiffi	Hund	[NULL]

nachher:

name	art	geschlecht
Fiffi	Hund	[NULL]
Bello	Hund	[NULL]

Allgemein lautet die Befehlsstruktur:

```
INSERT INTO Tabellenname (spaltenname1, spaltenname2, ...)
VALUES (wert_für_spalte_1, wert_für_spalte_2, ...)
```

Datensätze ändern mit UPDATE
Bereits in einer Tabelle vorhandene Datensätze können mittels eines UPDATE-Befehls angepasst werden:

```
UPDATE Haustier
SET geschlecht = "m"
WHERE name = "Bello"
```

Dieser Befehl sorgt dafür, dass bei allen Haustiereinträgen mit dem Namen Bello in der Spalte geschlecht der Wert "m" eingetragen wird.

vorher:

name	art	geschlecht
Fiffi	Hund	[NULL]
Bello	Hund	[NULL]

nachher:

name	art	geschlecht
Fiffi	Hund	[NULL]
Bello	Hund	m

Allgemein lautet die Befehlsstruktur:

```
UPDATE Tabellenname
SET spaltenname = neuer_wert
WHERE auswahlbedingung
```

Besonderes Augenmerk muss auf die Formulierung einer sinnvollen Auswahlbedingung gelegt werden, da alle Einträge der genannten Tabelle, auf die diese Bedingung zutrifft, angepasst werden. Ist die Auswahlbedingung nicht präzise genug, werden ggf. mehr Datensätze als beabsichtigt angepasst. Vergisst man die Auswahlbedingung komplett, werden alle Datensätze der genannten Tabelle geändert. Möchte man nur einen bestimmten Datensatz anpassen, ist es daher sinnvoll, als Auswahlbedingung den Wert der Primärschlüsselspalte(n) mit einem gesuchten Wert zu vergleichen.

Datensätze löschen mit DELETE
Mittels der allgemeinen Befehlsstruktur

```
DELETE FROM Tabellenname
WHERE auswahlbedingung
```

können vorhandene Datensätze aus der angegebenen Tabelle gelöscht werden. Der Befehl

```
DELETE FROM Haustier
WHERE art IS NOT NULL
AND geschlecht IS NULL
```

löscht beispielsweise alle Einträge in der Haustiertabelle, bei denen zwar die Art, aber kein Geschlecht angegeben ist.

vorher:

name	art	geschlecht
Fiffi	Hund	[NULL]
Bello	Hund	m

nachher:

name	art	geschlecht
Bello	Hund	m

Auch hier besteht grundsätzlich die Gefahr, dass durch eine zu ungenaue Auswahlbedingung mehr Datensätze gelöscht werden als beabsichtigt. Da Datenbanksysteme in der Regel keine „Rückgängig"-Funktion besitzen, mittels derer versehentlich gelöschte oder geänderte Daten leicht wiederhergestellt werden könnten, muss bei UPDATE- und DELETE-Befehlen von vornherein auf eine korrekte Formulierung geachtet werden. Vergisst man beim oben gezeigten DELETE-Befehl die letzte Zeile, wird beispielsweise unbeabsichtigt auch Bello aus der Tabelle entfernt:

```
DELETE FROM Haustier
WHERE art IS NOT NULL
```

vorher:

name	art	geschlecht
Fiffi	Hund	[NULL]
Bello	Hund	m

nachher:

name	art	geschlecht

Zugriff auf Dateien
Die Klassenstrukturen für den Dateizugriff sowie die vorhandenen Methoden und deren Parameter sind in den verschiedenen Programmiersprachen sehr unterschiedlich. Zwei Muster sind aber überall vorhanden.

1 **Lesen, Schreiben oder beides**
Die meisten Dateizugriffe sind „Lesen des gesamten Dateiinhalts" und „Schreiben des gesamten Dateiinhalts". Diese Zugriffsarten werden überall unterstützt und sind nicht nur für Dateien des Hintergrundspeichers verwendbar, sondern auch für sonstige Ein- und Ausgabeströme wie Tastatur, Terminalausgabe oder Verbindungen über das Internet. Für Dateien des Hintergrundspeichers wird in der Regel auch noch die Möglichkeit angeboten, abwechselnd zu schreiben und/oder zu lesen. Verbunden damit ist die Möglichkeit, die Schreib- / Leseposition innerhalb der Datei frei zu wählen.

2 **Text oder Binär**
Der zweite Aspekt ist die Interpretation des Dateiinhalts.
Bei binärer Interpretation werden die Bitmuster der Attributwerte direkt in die Datei geschrieben oder von dort gelesen. Das sichert eine 100%ige Korrektheit der wiederhergestellten Daten. Es setzt aber voraus, dass der gleiche Wert auf allen Rechnern, auf denen die Daten bearbeitet werden sollen, auch das gleiche Bitmuster hat. Diese Methode kann z. B. nicht verwendet werden, wenn auf einem Rechner ganze Zahlen mit 4 Byte, auf dem anderen Rechner mit 8 Byte, gespeichert werden. Sie funktioniert auch dann nicht, wenn auf dem einen Rechner zuerst das niederwertigste Byte einer Zahl geschrieben wird, auf dem anderen Rechner aber das höchstwertige Byte zuerst erwartet wird.
Bei der Speicherung der Information in Textform gibt es diese Probleme nicht. Auch die Verwendung beliebiger sprachabhängiger Zeichen ist heute kein Problem mehr, wenn man eine Unicode-Darstellung (UTF) verwendet. Am meisten wird die Darstellung UTF-8 verwendet, die in der Regel die kleinsten Dateigrößen ermöglicht. Allerdings benötigt diese Art der Speicherung fast immer mehr Platz als eine binäre Speicherung. Auch muss man bei der Speicherung von Kommazahlen in Kauf nehmen, dass mindestens das niederwertigste Bit vom Original verschieden wieder eingelesen werden kann.

Beispiel für die unterschiedlichen Strategien beim Dateizugriff: Lesen einer Textdatei

Java: Lesen zeilenweise

```
try
{
  BufferedReader eingabe = new
    BufferedReader(new
    FileReader(dateiName));
  while (true)
  {
    String zeile =
      eingabe.readLine();
    if (zeile == null)
    {
      break;
    }
    //Verarbeiten des Inhalts
  }
  eingabe.close();
}
catch (Exception e)
{
  //Fehlermeldung
}
```

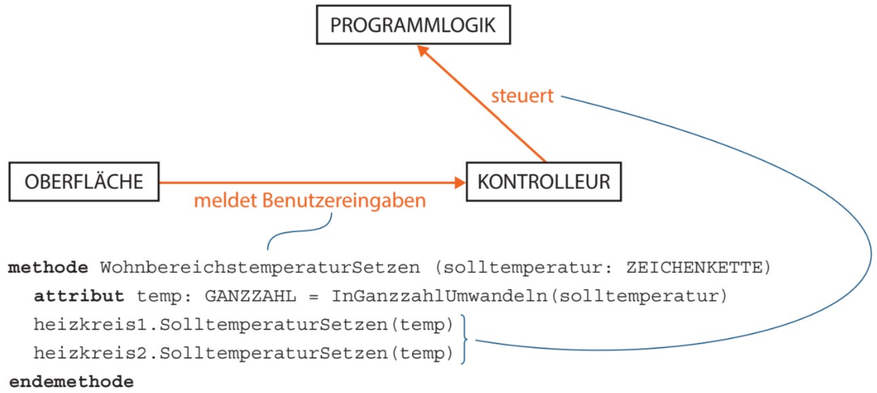
Swift: Lesen der Datei am Stück

```
do
{
  let inhalt = try
    String(contentsOfFile:
    datei, encoding: .utf8);
  let zeilen = inhalt.
    split(separator: "\n")
  for zeile in zeilen
  {
    //Verarbeiten des Inhalts
  }
}
catch
{
  //Fehlermeldung
}
```

6.6 Das Entwurfsmuster MVC

Möglichst wenig ändern
Viele Programme wie Textverarbeitung oder Tabellenkalkulation sind sowohl auf PCs und Laptops mit verschiedenen Betriebssystemen als auch auf Tablets und zum Teil auch auf Smartphones verfügbar. Dabei ist es nicht sinnvoll, für jede Anwendungssituation das Programm völlig neu zu entwickeln: der datenverarbeitende Teil (Programmlogik, Model) ist immer gleich, nur die Bedienoberfläche (Oberfläche, View) muss an die jeweilige Umgebung angepasst werden. Damit das einfach möglich ist, müssen Model und View möglichst gut voneinander getrennt sein.

Gut strukturiert ist halb gewonnen
Programmlogik und Oberfläche können natürlich nicht vollständig getrennt sein, da Eingaben in der Oberfläche von der Programmlogik verarbeitet werden müssen und das Ergebnis der Verarbeitung wieder in der Oberfläche dargestellt werden muss. Diese Schnittstelle zwischen Oberfläche und Programmlogik wird in einer einzigen Klasse isoliert, dem Kontrolleur (Controller); nur hier müssen dann Anpassungen vorgenommen werden, wenn z. B. die Oberflächenstruktur geändert wird.
Beispiel: In einer Smarthome-App gibt es einen Dialog zum Einstellen der Temperatur für den Wohnbereich; beim Drücken auf OK wird eine entsprechende Methode im Kontrolleur aufgerufen.




```
methode WohnbereichstemperaturSetzen (solltemperatur: ZEICHENKETTE)
  attribut temp: GANZZAHL = InGanzzahlUmwandeln(solltemperatur)
  heizkreis1.SolltemperaturSetzen(temp)
  heizkreis2.SolltemperaturSetzen(temp)
endemethode
```

Der Kontrolleur kann sich aber nicht darauf beschränken, Botschaften einfach nur weiterzureichen. Die Methode *WohnbereichstemperaturSetzen* muss z. B. überprüfen, ob die übergebene Zeichenkette überhaupt eine ganze Zahl darstellt oder die Zahl im zulässigen Bereich ist. Ist alles korrekt, wird der Dialog ausgeblendet, sonst muss er weiter angezeigt und um eine Fehlermeldung ergänzt werden. Dazu muss der Kontrolleur speichern, was die Oberfläche anzeigt, man sagt: in welchem Zustand sie ist.

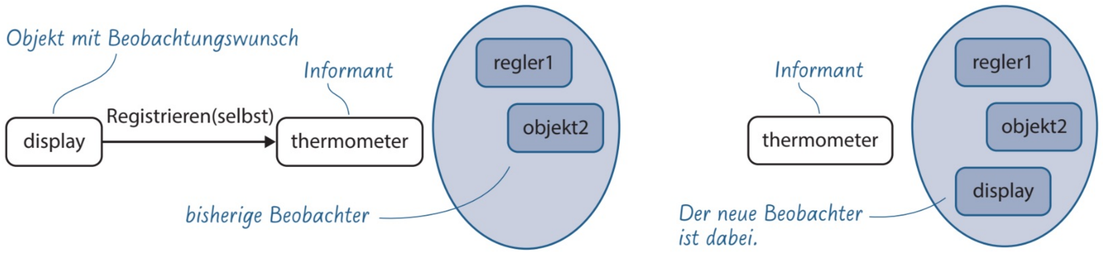
Oberfläche wird informiert

Die bisherige Struktur kann aber nicht alle Anforderungen erfüllen.

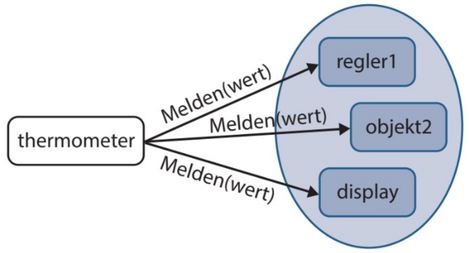
Bei einer Smart-Home-Steuerung muss z. B. ein Thermometer die aktuellen Temperaturwerte an mehrere Geräte melden, darunter die Anzeige der Temperatur im Raum in der Smarthone-App sowie das Regelgerät für die Fußbodenheizung. Auch dafür sollen der Erzeuger der Nachricht (das Thermometer) sowie die Beobachter (Display in der Smarthome-App, Regelgeräteansteuerung) nur das wirklich Notwendige voneinander kennen.

Damit ein Beobachterobjekt (Display, Heizungsregler) Nachrichten erhalten kann, muss es dem Erzeuger der Nachrichten zuerst mitteilen, dass es die Nachrichten auch erhalten möchte: Es muss sich registrieren.

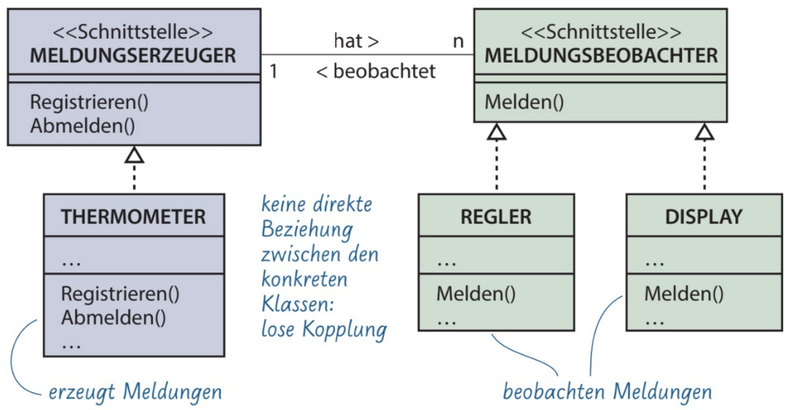
Im Bild links hat das Objekt thermometer (der Klasse THERMOMETER) bereits zwei Beobachter (regler1, objekt2; blau dargestellt). Das Objekt display (der Klasse DISPLAY) schickt eine Botschaft an thermometer mit dem Wunsch, in die Menge der Beobachter aufgenommen zu werden. Das Objekt thermometer erfüllt diesen Wunsch und nimmt display in die Menge der Beobachter auf (Bild rechts).



Jedes Beobachterobjekt muss dem Informanten eine Methode (oder mehrere Methoden, falls verschiedene Botschaften versendet werden sollen) als Schnittstelle anbieten, damit der Informant diese beim Eintreten einer Zustandsänderung (Temperatur hat sich geändert) aufrufen kann.



Um eine völlige Entkopplung zwischen den Beobachterklassen und der tatsächlichen Implementierung des Informanten zu erreichen, stellen sich nicht nur die Beobachter über eine Schnittstelle vor, auch der Informant selbst zeigt den Beobachtern über eine Schnittstelle nur die von diesen benötigten Methoden, nämlich *Registrieren* und – falls benötigt – das Gegenstück *Abmelden*.



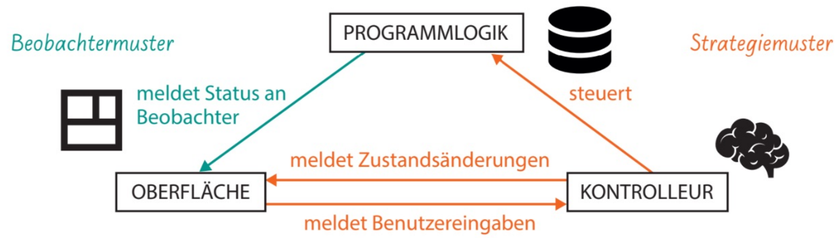
Das Bild auf der Vorseite macht deutlich, wie sich die verschiedenen Beobachter und Informanten hinter ihren Schnittstellen „verstecken" können. Nur die für den Nachrichtenaustausch wichtigen Methoden (*Melden* bzw. *Registrieren* und *Abmelden*) sind sichtbar. Die konkreten Klassen THERMOMETER, REGLER und DISPLAY nutzen nur die Schnittstellen, kennen sich aber gegenseitig nicht; sie sind lose gekoppelt.

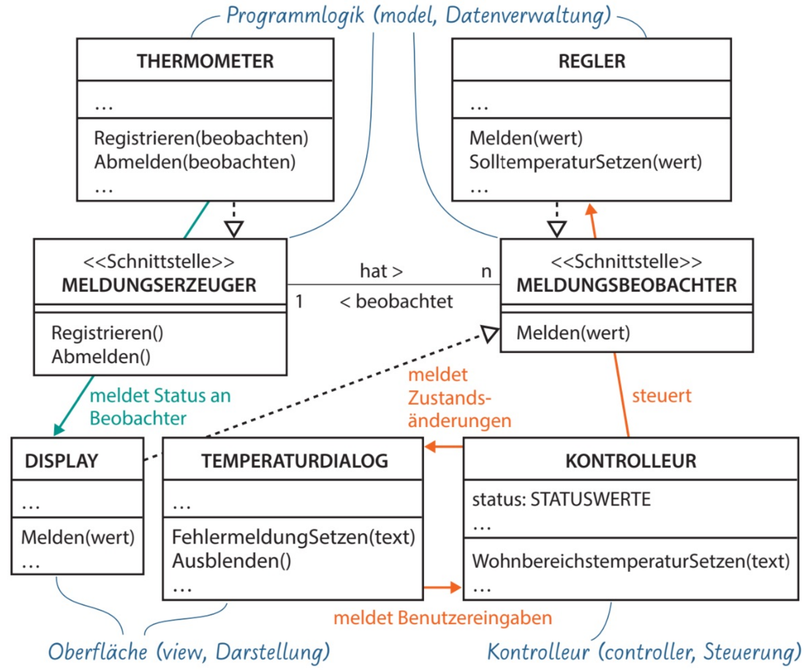
Die beschriebene Lösungsschablone nennt man Beobachtermuster.

Die fertige Struktur

Das Zusammenwirken der beschriebenen Teile hat den Namen **MVC** (für **model-view-controller**). Es gehört, wie das schon betrachtete Kompositum, zu dem bekannten und häufig verwendeten **Entwurfsmustern** (Architekturmuster, design patterns).

In der Abbildung unten wurde im Vergleich zu oben noch der Informationsfluss des Beobachtermusters mit grünen Pfeilen ergänzt. Das Registrieren der Beobachter ist hier nicht dargestellt.





6.7 Debuggen und Testen

```
ZeroDivisionError:
division by zero
```

```
Unterminated '/*' comment
```

```
reached end of file
while parsing
```

```
x = addiere(2,2)
print(x)
> 5
```

```
";" expected
```

```
java.lang.ArrayIndexOutOfBoundsException
```

→engl. bug: Wanze; gebräuchlich als Bezeichnung für Fehler

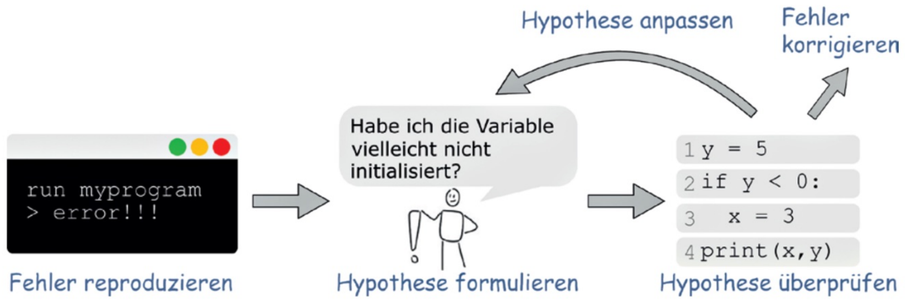
Fehler (→„Bugs“) sind ständiger Begleiter beim Programmieren; auch für Profis gehören sie zum Alltag. Dabei unterscheiden sich die Fehler hinsichtlich ihres Auftretenszeitpunktes, aber auch hinsichtlich ihrer Auswirkungen.

- **Kompilierzeitfehler** treten bereits zum Zeitpunkt der Übersetzung des Programms auf und liefern bereits Fehlermeldungen, die Hinweise zur genaueren Lokalisierung des Fehlers beinhalten.
- **Laufzeitfehler** treten erst bei Ausführung des Programms auf und haben dessen Abbruch und ebenfalls eine entsprechende Fehlermeldung zur Folge.
- **Logische Fehler** bringen das Programm zwar nicht zum Abbruch, führen jedoch zu falschen Ergebnissen bzw. Ausgaben.

Manchmal ist es hilfreich, seine Hypothesen explizit aufzuschreiben.

Debuggen

Beim **Debuggen** geht es darum, Fehler in einem Computerprogramm zu finden und zu beheben. Zunächst aufgestellte Hypothesen über die Ursache des Fehlers werden anhand des Quelltextes verifiziert bzw. verworfen.



Entwicklungsumgebungen bieten oft weitere Werkzeuge, die bei der Fehlerbehebung hilfreich sind (Objektinspektor, KI-Assistent).

Dabei können verschiedene Strategien angewendet und kombiniert werden.
Häufig bietet es sich an, die **Ausführung des Programmablaufs nachzuverfolgen**. Dabei geht es z. B. darum, herauszufinden, welchen Wert eine Variable zu einem bestimmten Zeitpunkt hat, oder ob ein Programmteil überhaupt ausgeführt wird. Konkret kann man dazu etwa einen **Schreibtischtest** durchführen, bei dem das Programm „per Hand“ Zeile für Zeile durchgegangen und die jeweils aktuellen Werte der Variablen notiert werden. Auch mit **textuellen Ausgaben** (etwa mittels Print-Befehlen oder Konsolenausgaben) lässt sich die aktuelle Belegung von Variablen ausgeben oder überprüfen, wie oft (oder ob überhaupt) eine bestimmte Anweisung

ausgeführt wird. Mit dem **Debugger** steht in den meisten Entwicklungsumgebungen zudem ein Werkzeug zur Verfügung, das das Nachvollziehen des Programmablaufs immens erleichtert. Die folgende Abbildung zeigt die wichtigsten Funktionalitäten eines Debuggers:

Der Debugger könnte anders aussehen, sollte aber über dieselben Funktionalitäten verfügen.

Um das Debuggen zu erleichtern, kann es hilfreich sein, den Code in kleinere Teile zu unterteilen und sie einzeln zu untersuchen. Durch das **Auskommentieren** bestimmter Codezeilen/Codeblöcke etwa werden diese übersprungen und es lässt sich leichter herausfinden, ob ein bestimmter Teil des Codes für das Problem verantwortlich ist.
Auch das **Testen** mit weiteren gezielt gewählten Eingaben, um das Verhalten des Programmes zu prüfen, kann Rückschlüsse über Art und Position des Fehlers liefern. Das hilft Hypothesen aufzustellen oder zu validieren.
Manchmal fällt das Aufstellen einer geeigneten Hypothese jedoch schwer oder der Fehler lässt sich nicht so einfach identifizieren. Dann sind auch das Sammeln weiterer Informationen oder die Suche nach Hilfe gute Ansatzpunkte. So kann man sich etwa bei Mitschülerinnen und Mitschülern oder in Entwicklerforen Hilfe suchen.

Beim ***rubberduck debugging*** verwendet man als Ansprechpartner eine Gummiente. Das bewusste Artikulieren wirkt manchmal Wunder … auch bei Profis.

Automatisiertes Testen

Einige Fehler werden bereits während des Programmierens und auch beim Debuggen bemerkt. Die Erfahrung zeigt aber, dass so längst nicht alle Fehler gefunden werden. Wichtig ist daher systematisches, automatisiertes Testen.

Mit automatisierten Tests können Entwicklerinnen und Entwickler eine große Anzahl von Tests in kurzer Zeit durchführen. Dies ist besonders bei großen Softwareprojekten, die im Team entwickelt werden, enorm nützlich, weil so für jede Änderung die Kompatibilität mit bestehenden Funktionalitäten geprüft wird. Da stets der exakt gleiche Testablauf durchgeführt wird, wird außerdem das Risiko menschlicher Fehler reduziert, was zu konsistenteren und zuverlässigeren Testergebnissen führen kann.

→ 6.8

Automatisierte Testfälle vereinfachen zudem das Refactoring, weil nach umfassenderen Umbauarbeiten am Code überprüft werden kann, ob die Funktionalität immer noch gegeben ist.
Im Gegensatz zum Debuggen hat das Testen lediglich das Ziel zu überprüfen, ob eine bestimmte Menge von Eingaben das gewünschte Verhalten in einem Programm verursacht. Je nachdem, ob die testende Person dabei Zugriff auf den Quelltext hat, wird zwischen **White-Box-** und **Black-Box-Tests** unterschieden.

- Bei **Black-Box-Tests** hat der Tester keine Kenntnis über die innere Struktur und die Implementierung der Software und testet lediglich die Funktionalität und Benutzbarkeit aus Sicht eines Nutzers.
- Bei **White-Box-Tests** hat der Tester Zugang zu allen Informationen über die Implementierung der Software. So lassen sich einzelne Funktionen oder Unterprogramme, die Integration verschiedener Teile der Software oder zusammenhängende Funktionalitäten, gezielt testen, um mögliche Fehler in der Implementierung aufzudecken.

Neben einzelnen Funktionalitäten wie Methoden oder Klassen in **Modultests** sollte auch das Zusammenspiel der verschiedenen Softwarekomponenten in **Integrationstests** geprüft werden. Damit kann sichergestellt werden, dass die Komponenten nicht nur isoliert, sondern auch als Gesamtsystem funktionieren.

Testbibliotheken wie JUnit bieten noch weitere Methoden zum Testen, z. B. um zu prüfen, ob eine Methode aufgerufen wurde.

Zum Testen stehen in jeder Programmiersprache verschiedene Bibliotheken zur Verfügung, die das Erstellen von Testfällen erleichtern.

Java

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import Taschenrechner;
import org.junit.jupiter.api.Test;

class MyFirstJUnitJupiterTests {

  private final Calculator taschenrechner = new Taschenrechner();

  @Test
  void testAddition() {
    assertEquals(2, taschenrechner.Addieren(1, 1));
  }

  @Test(expected = IllegalArgumentException.class)
  void testDivisionByZero () {
     Dividieren(2, 0);
  }
}
```

prüft, ob beide Eingabewerte nach der Auswertung gleich sind (also 2 == taschenrechner.Addieren(1,1))

Python

```
import unittest
import Addieren

class TestSum(unittest.TestCase):
    def test_addition (self):
        self.assertEqual(Addieren(1,1), 2, "1+1 sollte 2 ergeben")
    def test_division_by_zero(self):
        with self.assertRaises(ValueError):
            Dividieren(2, 0)

if __name__ == '__main__':
    unittest.main()
```

Swift

```
class AdditionsTest: XCTestCase {
    func testAddition() {
        XCTAssertEqual(Addieren(1,1), 2, "1+1 sollte 2 ergeben")
    }

    func testDivisionByZero() {
        XCTAssertThrowsError(try Dividieren(2,0)) { error in
            XCTAssertEqual(error.localizedDescription, "Divisor kann nicht 0
sein")
        }}}
```

Testen stellt dabei keinen formalen Nachweis für die Korrektheit eines Programms dar. Stattdessen wird versucht, das korrekte Verhalten der Software für möglichst viele erwartete, aber auch nicht erwartete Eingaben und Zustände zu überprüfen.

Im Wasserfallmodell steht das Testen typischerweise am Ende des Prozesses, in der agilen Entwicklung wird häufig auch nach dem sog. **Test-Driven-Development (TDD)** vorgegangen. Dabei werden Tests immer vor der eigentlichen Implementierung erstellt.

Selbst bei einfachen Programmen gibt es oft bereits derart viele Abläufe, dass ein vollständiger Beweis der korrekten Funktionalität praktisch unmöglich ist.

6.8 Softwarequalität aus Entwicklersicht

Softwarequalität geht über die reine Funktionalität hinaus
Die Qualität eines Softwareproduktes lässt sich nicht nur hinsichtlich ergonomischer Kriterien aus Sicht des Endanwenders beurteilen. Insbesondere bei größeren Softwareprojekten, die über längere Zeit gewartet und erweitert werden müssen, sollte neben der Anwenderperspektive auch die Entwicklersicht bei der Qualitätsbewertung berücksichtigt werden. Allgemein spricht man dabei auch von der **Wartbarkeit** eines Softwareproduktes.

Guter Code muss lesbar sein
Das vielleicht wichtigste Kriterium bei der Bewertung der Wartbarkeit ist die „Lesbarkeit" des Programmcodes. Dies bedeutet, dass sich Aufbau und Bedeutung des Codes auch von neu eingestellten Entwicklern oder solchen, die längere Zeit nicht mehr an diesem Projekt gearbeitet haben, schnell erschließen lassen. Eine zentrale Rolle spielt hierbei die Dokumentation des Programmcodes, aber auch die Verwendung aussagekräftiger und eindeutiger Namen für Klassen, Variablen und Methoden. Weiterhin sollten Formatierungskonventionen, etwa hinsichtlich Groß- und Kleinschreibung im gesamten Projekt einheitlich umgesetzt werden.

Checkliste Modularisierung:
- *Jede Klasse hat nur einen Zuständigkeitsbereich (Single-Responsibility-Prinzip).*
- *Jede Methode erfüllt genau eine Aufgabe und hat typischerweise unter 20 Zeilen.*

Modularer Aufbau statt Spaghetticode

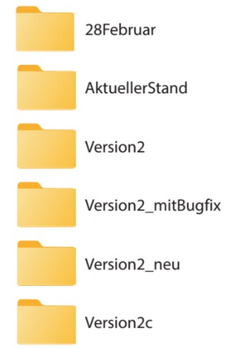
Mit dem Begriff „Spaghetticode" wird allgemein Programmcode bezeichnet, bei dem die Nachverfolgung von Programmabläufen mit Spaghetti auf einem Teller verglichen werden kann: Wie die Nudeln ist auch der Quellcode so komplex verwoben, dass die Nachverfolgung unübersichtlich ist und langwierige Abfolgen von einzelnen Codeabschnitten ergibt. Dies erschwert die Wartbarkeit enorm, da für nachträgliche Anpassungen des Programms stets der gesamte Code nachvollzogen werden muss und auch kleinere Fehlerbehebungen oder Funktionserweiterungen umfangreiche Codeanpassungen erforderlich machen können. Durch einen modularen Aufbau des Programms kann der Entstehung von Spaghetticode entgegengewirkt werden. Dabei wird die Programmfunktionalität sinnvoll in voneinander weitgehend unabhängige Einheiten aufgeteilt, die über klar definierte Schnittstellen miteinander interagieren.

Checkliste Refactoring:
- *sprechende Bezeichner*
- *„kleine" Klassen und Methoden*
- *Vererbung und Entwurfsmuster einsetzen*
- *Code-Duplikate vermeiden*

Refactoring bringt Ordnung in den Code
Die spätere Wartbarkeit des Softwareprodukts sollte beim Verfassen von Programmcode stets mit bedacht werden. Oft werden diesbezügliche Optimierungsmöglichkeiten aber auch erst im Nachhinein erkennbar. Beispielsweise kann es nach der Erweiterung einer bestehenden Klasse um zusätzliche Funktionalitäten sinnvoll erscheinen, die Klasse mit einem neuen Namen zu versehen, der den neuen Funktionsumfang besser beschreibt. Oder es wird nach Erweiterungen erkannt, dass eine Klasse nunmehr nicht nur eine Aufgabe erfüllt, sondern mehrere, und deswegen aufgeteilt werden sollte. Kommen dagegen bestimmte Abschnitte mehrfach in einer Klasse vor, sollten diese in eine Methode ausgelagert werden. Das Vermeiden dieser Redundanz in Form von Code-Duplikaten erhöht die Wartbarkeit deutlich. Mit dem Begriff **Refactoring** werden allgemein Anpassungen am Programmcode beschrieben, die lediglich der Verbesserung der Wartbarkeit dienen und die eigentliche Programmfunktionalität nicht verändern. Viele Entwicklungsumgebungen bieten hierfür spezielle Werkzeuge an, die z. B. beim Umbenennen einer Klasse automatisch dafür sorgen, dass ihr Name überall im Code einheitlich geändert wird.

6.9 Versionsverwaltungssysteme

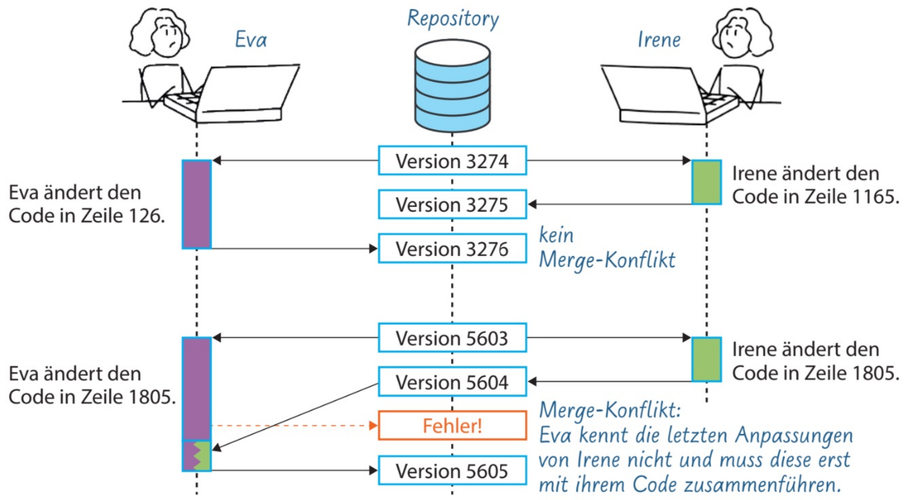
Den Überblick behalten – auch bei größeren Projekten
Kaum ein Softwareprojekt lässt sich „in einem Rutsch" vollständig und fehlerfrei realisieren. In der Regel entsteht ein Programm daher schrittweise und wird von Version zu Version verbessert und ggf. erweitert. Dabei kann es leicht zu einem Versionschaos kommen, wie es in der nebenstehenden Abbildung angedeutet ist. Oft arbeiten zudem mehrere Personen an einem Projekt, sodass der jeweils aktuelle Stand des Programmcodes immer wieder unter allen Beteiligten ausgetauscht werden muss. Auch dabei kann leicht ein Durcheinander entstehen, wenn etwa zwei Personen gleichzeitig den Code bearbeiten und es so am Ende keinen einheitlichen„aktuellen Stand" des Projektes mehr gibt.



Versionsverwaltungssysteme schaffen Ordnung
Um die genannten Probleme reduzieren zu können, wurden sogenannte Versionsverwaltungssysteme entwickelt. Ein zentrales Konzept ist dabei das sogenannte →**Repository**. In diesem werden verschiedene Versionsstände des Programmcodes automatisiert in einer hierarchischen Struktur archiviert. Dadurch kann der zeitliche Verlauf der Codeentwicklung leicht nachvollzogen werden und auch der Zugriff auf ältere Versionen bleibt möglich. Ältere Versionsverwaltungssysteme sehen meist nur ein zentrales Repository vor. Alle Entwickler können sich den aktuellen Codestand von dort herunterladen („check out") und nach getaner Arbeit ihre Änderungen wieder hochladen („commit"). Arbeiten mehrere Entwickler gleichzeitig, kann es dabei zu einem sogenannten →Merge-Konflikt kommen, wenn verschiedene Entwickler den gleichen Codeabschnitt auf unterschiedliche Weise verändert haben. In diesem Fall muss dann manuell entschieden werden, welche Variante als neuer Versionsstand ins Repository einfließt bzw. wie man die Varianten zusammenführt.

→engl. repository: Lager, Depot; selten auch lat. „Repositorium"

→engl. merge: zusammenführen, vereinigen



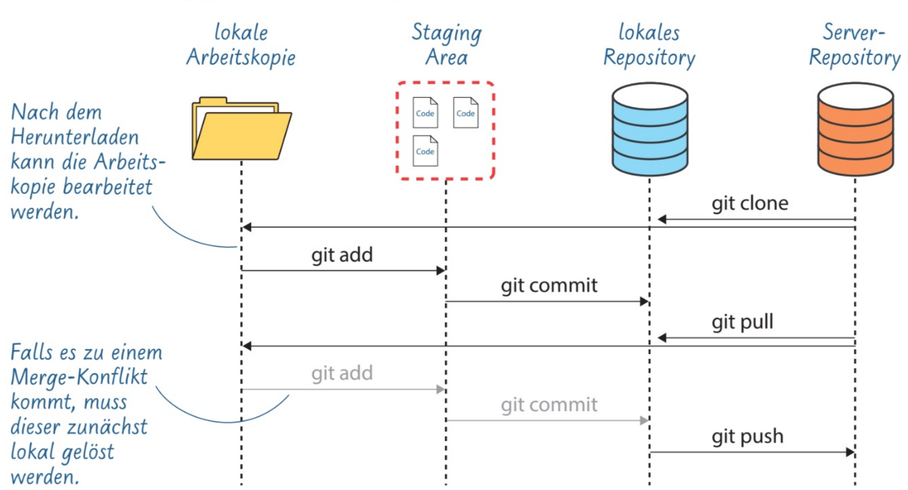
→Linus Torvalds (*1969), „Erfinder" des Betriebssystems Linux

Verteilte Versionsverwaltung mit git

Bei neuen Projekten kommt heute als Versionsverwaltungssystem fast immer das von →Linus Torvalds erfundene und frei verfügbare **git** zum Einsatz, welches eine dezentrale Versionsverwaltung erlaubt. Dabei besitzt jeder Entwickler ein **lokales Repository** zur Versionsverwaltung, welches mit anderen Repositories synchronisiert werden kann. Oft gibt es ein zentrales **Server-Repository**, in welchem das Projekt verwaltet wird. Der Arbeitsablauf ist dann wie folgt:

Idealerweise können die git-Befehle direkt innerhalb deiner Entwicklungsumgebung aufgerufen werden. Teilweise ist dort die Staging Area nicht sichtbar, der add-Befehl in diesem Fall nicht nötig.

- Mittels des *clone*-Befehls kann eine lokale Kopie des Server-Repository auf dem eigenen Rechner erzeugt werden. Dabei wird zusätzlich auch gleich eine **lokale Arbeitskopie** des aktuellen Projektstandes angelegt. Alternativ kann mit dem *init*-Befehl ein neues leeres Repository angelegt werden.
- Die lokale Arbeitskopie des Codes kann nach Belieben bearbeitet werden. Da nicht immer alle geänderten Dateien sofort in die nächste Projektversion einfließen sollen, müssen diese mittels *des add*-Befehls explizit in einen Vorbereitungsbereich (engl. **staging area**) aufgenommen werden.
- Mit dem *commit*-Befehl wird auf Basis der Änderungen im Vorbereitungsbereich eine neue Projektversion im lokalen Repository angelegt.
- Bevor man nun die lokalen Änderungen ins Server-Repository hochlädt, sollte man über einen *pull*-Befehl mögliche Änderungen auf dem Server-Repository herunterladen und mit dem lokalen Repository zusammenzuführen. Dabei kann es zu Merge-Konflikten kommen, falls zwischenzeitlich andere Entwickler ihre Codeänderungen in das Server-Repository eingepflegt haben. Das Versionsverwaltungssystem versucht in diesem Fall zunächst, die Konflikte automatisch zu lösen. Nur wenn nicht eindeutig erkennbar ist, welche Codebestandteile übernommen werden sollen, ist ein manuelles Eingreifen erforderlich.
- Anschließend kann der neue „aktuelle Stand" des Projektes mit dem *push*-Befehl an das Server-Repository gesendet und dort gespeichert werden.



6.10 Projektvorschläge

Die Vielfalt der heutzutage eingesetzten Software ist groß. Entsprechend groß ist auch die Auswahl an möglichen Projekten. Im Folgenden finden sich einige Ideen für Projekte. Bitte klären Sie allgemeine Vorgaben mit Ihrer Lehrkraft. Typische Vorgaben sind:

- Das Programm enthält eine grafische Benutzeroberfläche.
- Eine persistente Datenspeicherung ist umgesetzt.
- Das Team, typisch sind sechs bis acht Mitglieder, organisiert sich selbst und achtet dabei auf Transparenz und Reflexion des Arbeitsprozesses sowie eine aussagekräftige Dokumentation und eine wertschätzende Kommunikation.
- Es wird auf Softwarequalität aus Nutzer- und Entwicklersicht sowie auf Tests geachtet.
- Entwurfsmuster, Bibliotheken und ein Versionsverwaltungssystem werden eingesetzt.
- Planen Sie mit einer Basisversion und Erweiterungsmöglichkeiten.

Sie halten eine andere Programmierumgebung für die Projektumsetzung sinnvoll? Reden Sie mit Ihrem Team und der Lehrkraft – es muss für alle passen!

1 Spielerische Erlebnisse schaffen

- **a** Brettspiel wie Schach, Reversi oder Hnefatafl
- **b** Digitaler Escape-Room
- **c** 2D-Platformer: Spiel, bei dem man einen Charakter steuern muss, der z. B. Hindernisse überwindet und Gegner besiegt
- **d** Multiplayer-Quiz: Quizspiel mit mehreren Clients, bei dem die Spielenden gegeneinander antreten und Fragen beantworten müssen
- **e** Dunkles Labyrinth: In einem Labyrinth sieht man nur die unmittelbaren Nachbarzellen und muss zum Ausgang finden.
- **f** Pong über mehrere Geräte: Das Spiel Pong wird so gespielt, dass jedes Gerät nur das halbe Spielfeld sieht. Bei Verlassen des einen Displays wird der Ball auf dem anderen Spielfeld angezeigt.
- **g** Umsetzung eines einfachen Kartenspiels (z. B. Mau-Mau, Grasoberln, etc.). Jeder Teilnehmer spielt dabei an einem eigenen Gerät; der Spielzustand wird in einer gemeinsamen Datenbank verwaltet.

2 Verwaltung vereinfachen

- **a** Projekt- und Mitarbeiterverwaltungssoftware für die Schülermitverantwortung
- **b** Nachhilfevermittlung: System, das Nachfrage und Angebote zur Nachhilfe zusammenbringt
- **c** Buchungssystem für Co-Working-Space: Anwendung, die es ermöglicht, Tische (Räume, Technik) in einem Co-Working-Space für bestimmte Zeitfenster zu buchen
- **d** Inventarverwaltung der Schulbibliothek: System, das es Bibliotheksmitarbeitenden ermöglicht, Bücher und Benutzerdaten zu verwalten

	Montag 02. Oktober 2023			
Platz	A	B	C	A
08:00 bis 09:00 Uhr	Geschlossen	Frei	Frei	Geschlossen
09:00 bis 10:00 Uhr	Geschlossen	Frei	Abo	Geschlossen
10:00 bis 11:00 Uhr	Frei	Frei	Frei	Frei
11:00 bis 12:00 Uhr	Abo	Frei	Frei	Abo
12:00 bis 13:00 Uhr	Abo	Abo	Frei	Abo
13:00 bis 14:00 Uhr	Abo	Abo	Frei	Abo

Beachten Sie bei der Umsetzung der Projekte Aspekte des Datenschutzes und Urheberrechts!

Es gibt im Internet Schnittstellen (APIs), über die man Daten herunterladen kann. Einmal implementiert, stehen den Usern nach jedem Programmstart die aktuellen Daten zur Verfügung.

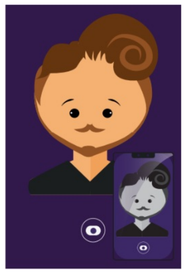
3 Praktische Begleiter schaffen

- **a** Focus-Tracker: Programm, das das einfache Erfassen von Zeit auf verschiedene Aktivitäten (Lernen für das Englischabitur, Lernen für das Informatikabitur, …) erlaubt
- **b** Abischnitt-Rechner: Nach Eingabe der Belegungsdaten kann man die eigenen Noten eintragen und jederzeit eine Prognose für den Abiturschnitt berechnen.
- **c** Vokabeltrainer, der auch zu jeder Vokabel einen Beispielsatz enthält: Wählbare Kategorien wie unregelmäßige Verben, Kontexte, Werte, Umwelt, Politik können hilfreich für das Lernen sein.
- **d** Kochbuch mit Zusammenstellung einer personenzahlabhängigen Einkaufsliste
- **e** Wettprogramm für eine Sportveranstaltung: Mehrere Mitspielende geben Tipps ab und auf Basis der Ergebnisse wird eine Rangliste ermittelt.
- **f** Wettervorschau: Anwendung, die Nutzerinnen und Nutzern übersichtlich das aktuelle bzw. vorhergesagte Wetter präsentiert und Statistiken (wie Sonnenscheindauer) zum ausgewählten Ort anzeigen kann
- **g** Nachhaltigkeits-Tracker: App, mit der tägliche Aktivitäten wie Essen, Transport oder Freizeitaktivität erfasst werden können, um deren Energieverbrauch und CO_2-Emission zu ermitteln; durch entsprechende Analyse und Visualisierung, kann sie ein nachhaltigeres Verhalten unterstützen.
- **h** Notizbuch/To-Do-Listen-App: Programm, das es dem User ermöglicht, Notizen und To-Do-Listen zu erstellen und zu verwalten

4 Anwendungen der Informatik

- **a** Simulation mit änderbaren Parameterwerten zum Erkenntnisgewinn für reale Vorgänge: Räuber-Beute-Beziehung (z. B. Wechselbeziehung der Populationen von Füchsen und Hasen), Autoverkehr (Optimierung von Ampelschaltungen), Wachstumsvorgänge (Conways Spiel des Lebens als biologischer Mikrokosmos), …
- **b** Bioinformatik: Programm zu einer einfachen Auswertung von DNA-Sequenzvergleichen beispielsweise zur Untersuchung der Abstammungs- und Verwandschaftsbeziehungen von Lebewesen.
- **c** Musik und Informatik: Anwendung zur Speicherung und Manipulation von Melodien z. B. im MusicXML-Format.

5 Mit Dateien arbeiten



- **a** Fotofilter; Bildkompression: Programm, das verschiedene Veränderungen von Pixelgrafiken ermöglicht wie Veränderung der Farbtöne, schwarz-weiß, Farbersetzung, Morphing, Kostümierung, Alterung, Kompression
- **b** Programm zum Erstellen und Verändern von Vektorgrafiken
- **c** Daten (aus Textdateien) werden eingelesen, visualisiert und ausgewertet, z. B. EKG-Messungen, Statistiken der Weltgesundheits- oder einer Umweltorganisation, …
- **d** Datei-Hosting-System: Ein System, das es Usern ermöglicht, Dateien hoch- und herunterzuladen und diese mithilfe eines Links mit anderen zu teilen

6 Mit mehreren Geräten arbeiten

- **a** Abstimmungstool, das z. B. in der Schule oder bei Veranstaltungen eine unkomplizierte Befragung der Anwesenden ermöglicht und die Ergebnisse anschließend sinnvoll visualisiert.
- **b** Netzwerkscanner, der lokale Netzwerke nach verfügbaren Geräten durchsucht, deren Namen, IP- und Mac-Adresse listet und auf zur Verfügung gestellte Dienste (etwa http) zugreifen kann.

Lösungen zu „Teste dich selbst“

Aufgaben in Kapitel 1

T1 Richtig oder falsch?

- **a** wahr
- **b** Falsch: Die Warteschlange hat eine rekursive Datenstruktur, aber noch keine rekursiven Methoden.
- **c** Falsch: Die Trennung von Struktur und Daten geschieht durch Aufteilung in die konkrete Anwendungsklasse (z. B. PATIENT) und KNOTEN.
- **d** wahr
- **e** Falsch: Stapel finden breite Anwendung, z. B. in der Lagerverwaltung oder auch beim Aufrufstapel der Methoden.

T2 Ich check’s, dank deiner Hilfe!

Am besten, Sie erklären es Ferdi, indem Sie mit ihm zusammen das Buch durchblättern: Liste (Kapitel 1.4), Warteschlange (1.1), Trennung von Struktur und Daten (1.2), rekursive Methode (1.4), Entwurfsmuster Kompositum (1.5), Stapel (1.6).

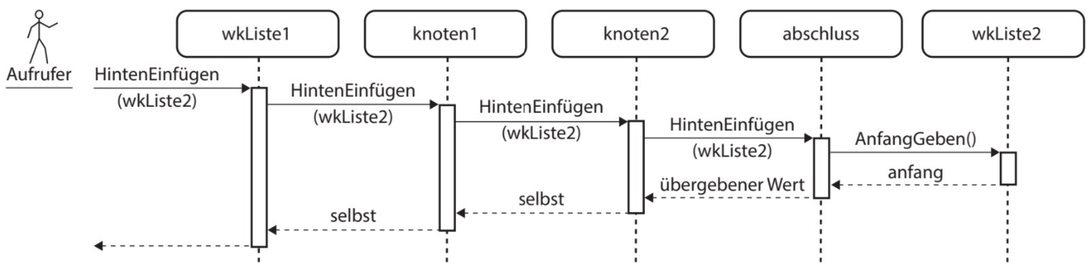
T3 Schulverwaltungssoftware (aus Abitur 2022)

- **a** Das Entwurfsmuster Kompositum erspart zahlreiche bedingte Anweisungen zur Prüfung, ob die Liste leer ist bzw. ein Knoten einen Nachfolger hat. Durch die Klassen KNOTEN (Verwaltung der Vorgänger-Nachfolger-Beziehung) und WAHLKURS (konkrete Anwendungsklasse) ist die Trennung von Struktur und Daten realisiert, auch wenn durch Verwendung einer Schnittstelle noch mehr Unabhängigkeit erreicht werden könnte.
- **b** Im Pseudocode des Buches, der im Sinne des Abiturs keine wählbare Programmiersprache darstellt:

```
klasse WAHLKURSLISTE
    anfang: LISTENELEMENT
  methode AnzahlKurseGeben(fachgebiet: ZEICHENKETTE) -> GANZZAHL
    return anfang.KurseZählen(fachgebiet)
  endemethode
  methode ZuKleineKurseLöschen(minTNZahl: GANZZAHL)
    anfang = anfang.ZuKleineKurseLöschen(minTNZahl)
  endemethode
endeklasse

klasse LISTENELEMENT {abstract}
  methode KurseZählen(fachgebiet: ZEICHENKETTE)->GANZZAHL {abstract}
  methode ZuKleineKurseLöschen(minTNZahl: GANZZAHL)-> LISTENELEMENT ↵
                                                              {abstract}
endeklasse
```

```
klasse KNOTEN beerbt LISTENELEMENT
  nachfolger: LISTENELEMENT
  daten: WAHLKURS
  methode KurseZählen(fachgebiet: ZEICHENKETTE) -> GANZZAHL
          anzahl: GANZZAHL
          anzahl = nachfolger.KurseZählen(fachgebiet)
          falls daten.FachgebietGeben()==fachgebiet
          dann
                  anzahl = anzahl + 1
          endefalls
          return anzahl
  endemethode
  methode ZuKleineKurseLöschen(minTNZahl: GANZZAHL) -> LISTENELEMENT
          nachfolger = nachfolger.ZuKleineKurseLöschen(minTNZahl)
          falls daten.TNZahlGeben()<minTNZahl
          dann
                  return nachfolger
          sonst
                  return selbst
          endefalls
  endemethode
endeklasse

klasse ABSCHLUSS beerbt LISTENELEMENT
  methode KurseZählen (fachgebiet: ZEICHENKETTE) -> GANZZAHL
          return 0
  endemethode
  methode ZuKleineKurseLöschen(minTNZahl: GANZZAHL) -> LISTENELEMENT
          return selbst
  endemethode
endeklasse
```

c



T4 Burning Man (aus Abitur 2021)

a In VERANSTALTUNGSLISTE:

```
methode SortiertEinfuegen(vNeu: VERANSTALTUNG)
  anfang = anfang.SortiertEinfuegen(vNeu)
endemethode
methode AnzahlGeben(tag: GANZZAHL, zeitfenster: GANZZAHL)->GANZZAHL
  return anfang.AnzahlGeben(tag, zeitfenster)
endemethode
```

In LISTENELEMENT:

```
methode SortiertEinfuegen(VERANSTALTUNG vNeu)-> LISTENELEMENT {abstract}
methode AnzahlGeben(tag: GANZZAHL, zeitfenster: GANZZAHL)->GANZZAHL ↵
                                                              {abstract}
```

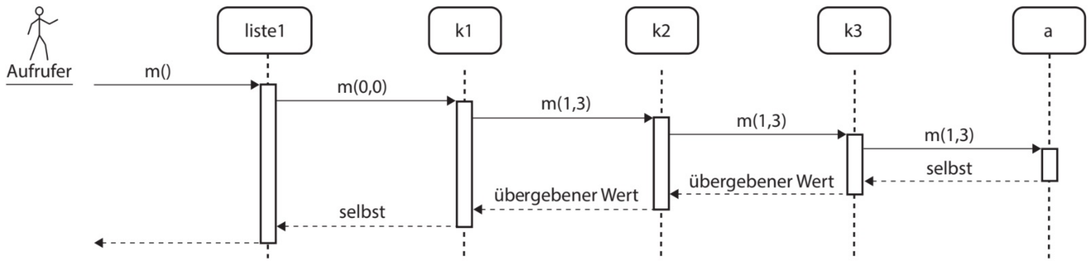
In KNOTEN:

```
konstruktor(vNeu: VERANSTALTUNG, lNeu: LISTENELEMENT)
  daten = vNeu
  nachfolger = lNeu
endekonstruktor
methode SortiertEinfuegen(VERANSTALTUNG vNeu)-> LISTENELEMENT
  falls daten.TagGeben()>vNeu.TagGeben()↵
                oder (daten.TagGeben()==vNeu.TagGeben()↵
                und daten.ZeitfensterGeben()> vNeu.ZeitfensterGeben())
  dann
        return neu KNOTEN (vNeu, selbst)
  sonst
        nachfolger = nachfolger.SortiertEinfuegen(vNeu)
        return selbst
  endefalls
endemethode
methode AnzahlGeben(tag: GANZZAHL, zeitfenster: GANZZAHL) -> GANZZAHL
  falls tag==daten.TagGeben() und zeitfenster==daten.ZeitfensterGeben()
  dann
        return nachfolger.AnzahlGeben(tag, zeitfenster) + 1
  sonst
        return nachfolger.AnzahlGeben(tag, zeitfenster)
  endefalls
endemethode
```

In ABSCHLUSS:

```
methode SortiertEinfuegen(VERANSTALTUNG vNeu)-> LISTENELEMENT
  return neu KNOTEN(vNeu, selbst)
endemethode
methode AnzahlGeben(tag: GANZZAHL, zeitfenster: GANZZAHL) -> GANZZAHL
  return 0
endemethode
```

b



Sind mehrere Veranstaltungen am gleichen Tag und im gleichen Zeitfenster in der Liste, so entfernt die Methode jeweils alle Veranstaltungen außer der ersten aus der Liste.

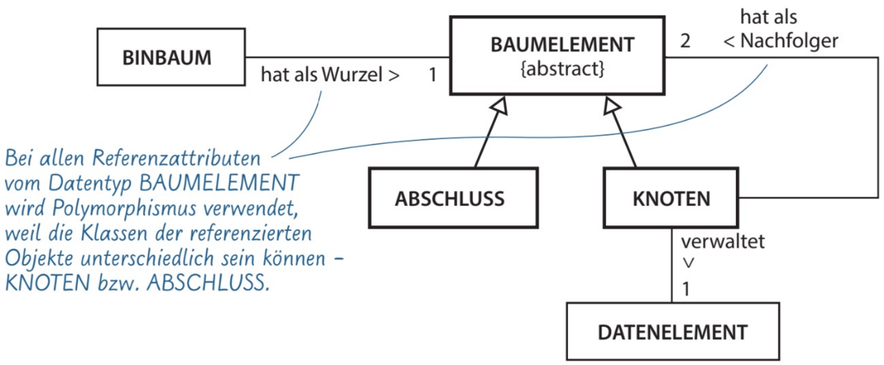
Aufgaben in Kapitel 2

T1 Rekursive Datenstruktur Baum und Entwurfsmuster Kompositum

a Ein geordneter Binärbaum ist eine effiziente Datenstruktur für die Suche in großen Datenbeständen, z. B. die Nutzerinnen und Nutzer eines Onlineshops.
Mit der Datenstruktur Baum können Anwendungsaufgaben wie der Aufbau einer hierarchischen Struktur oder die Kompression von 2D-Grafiken über den Quad-Tree umgesetzt werden. Eine rein linearen Datenstruktur (Liste) wäre dafür nicht ausreichend.

b Da $2^8 = 256 < 500 < 2^9 = 512$ sind mindestens 9 Ebenen nötig. Bei entarteten Bäumen kann die Ebenenanzahl jedoch deutlich höher liegen.

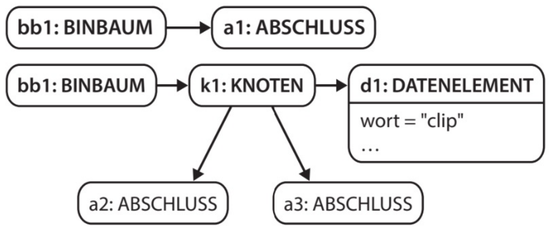
c, e



Die Datenstruktur ist rekursiv, weil von der Klasse KNOTEN (die auch ein BAUMELEMENT ist), eine Beziehung zur Klasse BAUMELEMENT existiert.

d Die Ordnung des Binärbaums wird durch die Methode *Einfügen* sichergestellt. Aus dem Klassendiagramm lässt sich jedoch nicht erkennen, wie diese Methode arbeitet.

f

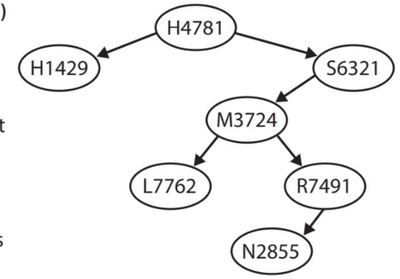


g Im Rumpf der Methode *Einfügen(DATENELEMENT)* des Objekts binbaum wird die Methode *Einfügen* des Anfangs aufgerufen. Der Anfang im leeren Baum ist ein Abschluss (hier a1 genannt). Der Abschluss erzeugt einen Knoten, der das eingegebene Datenelement (hier d1) referenziert sowie zwei Abschlüsse als Nachfolger hat. Die Referenz auf den neuen Knoten (hier k1 genannt) gibt er dem Aufrufer als Antwort zurück. Das Binärbaum-Objekt setzt die erhaltene Referenz als neuen Anfang. (a1 selbst ist i. d. R. nicht mehr Teil der Datenstruktur und wurde deshalb im unteren Objektdiagramm weggelassen.)

T2 Ich check's, dank deiner Hilfe!

Am besten, Sie erklären es Ferdi, indem Sie mit ihm zusammen das Buch durchblättern: Bestandteile und Eigenschaften des Baums (Kapitel 2.1), rekursive Methoden *Einfügen* und *Suchen* (2.2 bzw. 2.3), rekursive Methoden *PreorderAusgeben, InorderAusgeben, PostorderAusgeben* und deren Einsatzgebiete (2.4).

T3 Kundenverwaltung Spieleverleih (aus Abitur 2020)



a H4781, H1429, S6321, M3724
H4781, S6321, H1429, M3724
H4781, S6321, M3724, H1429.

b Eine nach dem Schlüssel sortierte Ausgabe ist mit dem Inorder-Durchlauf möglich. Hierbei wird zuerst die Methode vom linken Nachfolger rekursiv aufgerufen, dann der eigene Wert ausgegeben und zuletzt die Methode des rechten Nachfolgers rekursiv aufgerufen.
Der Baum ist nach der eindeutigen Kundennummer sortiert, in der Regel stimmt diese Sortierung nicht mit einer Sortierung nach dem Nachnamen überein.

c Für die Anzahl n der Ebenen gilt:

$2^n - 1 >= 2500 \quad | +1$
$2^n >= 2501 \quad | \log 2$
$n >= \log_2(2501) \Rightarrow n >= 11{,}2$ Es müssen mindestens 12 Ebenen sein.

d Ein balancierter Baum hat den Vorteil, dass die Ebenenanzahl minimal ist und somit der Baum schnell durchsucht werden kann. Ein Nachteil ist jedoch der hohe Verwaltungsaufwand beim balancierten Einfügen.

e siehe Klassendiagramm im Merkkasten S. 65 – ersetzen Sie die Klasse DATENELEMENT durch die Klasse KUNDE sowie BINBAUM durch KUNDENBAUM

f Im Pseudocode des Buches, im Sinne des Abiturs keine wählbare Programmiersprache

Klasse KUNDENBAUM

```
methode passwortPruefen(kdnr: ZEICHENKETTE, pw: ZEICHENKETTE)
                                                    -> WAHRHEITSWERT
    return wurzel.passwortPruefen(kdnr, pw)
endemethode
```

Klasse BAUMELEMENT **{abstract}**

```
methode passwortPruefen(kdnr: ZEICHENKETTE, pw: ZEICHENKETTE) {abstract}
                                                    -> WAHRHEITSWERT
```

Klasse KNOTEN:

```
methode passwortPruefen(kdnr: ZEICHENKETTE, pw: ZEICHENKETTE)
                                                    -> WAHRHEITSWERT
        attribut vergleich: GANZZAHL
        vergleich = inhalt.kundennummerVergleichen(kdnr)
```

```
            falls vergleich ==0
            dann
                    return inhalt.istPasswortGleich(pw)
            sonst
                    falls vergleich <0
                    dann
                            return nachfolgerLinks.passwortPruefen(kdnr, pw)
                    sonst
                            return nachfolgerRechts.passwortPruefen(kdnr,pw)
                    endefalls
            endefalls
    endemethode
```

Klasse ABSCHLUSS:

```
    methode passwortPruefen(kdnr: ZEICHENKETTE, pw: ZEICHENKETTE)
                                                        -> WAHRHEITSWERT
      return FALSCH
    endemethode
```

Aufgaben in Kapitel 3

T1 Richtig oder falsch?

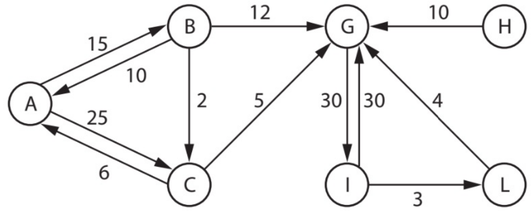
a richtig
Es kann auch mit „falsch“ argumentiert werden, weil bei verschränkter Rekursion keine Methode „sich“ selbst aufrufen muss.
b falsch: Ohne Abbruchbedingung wird die Rekursion nie beendet.
c falsch: z. B. verzweigte Rekursion
d richtig
e richtig
f falsch: Bei der Tiefensuche wird zuerst in die Tiefe des Graphen gegangen.
g falsch: Durch fortgesetzte Tiefensuche wird auch die optimale Lösung gefunden.

T2 Ich check’s, dank deiner Hilfe!

Beispiele können aus dem Lehrtext gewählt werden.
Bei der **Rekursion** rufen Methoden sich selbst auf **(rekursiver Aufruf)**. Eine **Abbruchbedingung** sorgt dafür, dass die rekursiven Aufrufe zu einem Ende kommen; damit ist das **Rekursionsende** erreicht. Bei der **linearen Rekursion** ruft sich eine rekursive Methode an genau einer Stelle selbst auf, bei **verzweigter Rekursion** an zwei oder mehr Stellen.
Das **Backtracking**-Verfahren ist ein rekursives Verfahren, das schrittweise auf eine Lösung zu steuert und durch die rekursiven Aufrufe auf seiner eigenen Spur zurückgeht, wenn der Weg in eine Sackgasse geraten ist. Wird das Backtracking nicht beim Finden der ersten Lösung beendet, sondern fortgesetzt, bis alle Lösungen gefunden sind, wird auch die optimale Lösung gefunden.
Die **Tiefensuche** ist ein Backtracking-Verfahren, das auf einen Graphen zum Besuch der Knoten angewendet wird. Die **optimale Lösung** ist dabei in der Regel **der kürzeste Weg** von einem Startknoten zum Zielknoten.

T3 Programmieren

a Der Graph ist gerichtet, weil die Adjazenzmatrix nicht symmetrisch zur Diagonale ist.



b

```
methode KnotenBesuchen(aktuell: GANZZAHL)
  besuchteKnoten.Hinzufügen(aktuell)
  zähle nummer von 0 bis knoten.ElementanzahlGeben() - 1
      falls (matrix.ElementGeben(aktuell).ElementGeben(nummer) >= 10↵
                                    und (NICHT besuchteKnoten.Enthält(nummer))
      dann
        KnotenBesuchen(nummer)
      endefalls
  endezähle
endemethode
```

A, B, G, I, C

Aufgaben in Kapitel 4

T1 Richtig oder falsch?

a falsch: Die Ausführungsstränge nebenläufiger Programme können (müssen aber nicht) parallel ausgeführt werden. Ob die Ausführung tatsächlich parallel abläuft, hängt von der verfügbaren Hardware und dem Betriebssystem ab.
b richtig
c richtig
d falsch: Eine Verklemmung kann nur auftreten, wenn parallele Threads mehr als eine Ressource gemeinsam nutzen.
e falsch: Der Thread kann erst dann wieder aktiv werden, wenn ein anderer Thread *Benachrichtigen()* aufgerufen hat.
f richtig: Aber es genügt bereits, dafür zu sorgen, dass eine der vier Coffman-Bedingungen nicht erfüllt ist.

T2 Ich check’s, dank deiner Hilfe!

a **Thread:** Ein Thread definiert einen Ausführungsstrang, der unabhängig von anderen Threads nebenläufig ausgeführt werden kann. Die Threads eines schwergewichtigen Prozesses können jedoch auf einen gemeinsamen Speicherbereich zugreifen. (Umsetzung im Programmcode siehe S. 119).
Monitor: Ein Monitor ist ein von einigen Programmiersprachen bereitgestelltes Konstrukt, welchen Zugriff mehrerer Threads auf einen kritischen Abschnitt synchronisiert. (Umsetzung im Programmcode siehe S. 124).
Warten und Benachrichtigen: Ein Thread kann einen Monitor durch den Aufruf von *Warten()* freigeben. Ruft ein anderer Thread *Benachrichtigen()* auf, hat sich üblicherweise der Zustand der Ressource in der Wartebedingung der Wartewiederholung verändert, und der benachrichtigte Thread kann den Monitor nun betreten.
Beispiele zur Anwendung in den verschiedenen Programmiersprachen findet man in den jeweiligen Kapiteln oder in den bearbeiteten Projekten.

b **Verklemmungen verhindern:** Verklemmungen können verhindert werden, indem man dafür sorgt, dass mindestens eine der vier Coffman-Bedingungen (wechselseitiger Ausschluss, Halten und Warten, Ununterbrechbarkeit und zyklisches Warten) nicht zutrifft.

T3 Schwierigkeiten beim Tabletverleih

a Beispielszenario: Ist nur noch ein Gerät vorhanden und wird die Methode *GerätAusleihen* von zwei Threads nahezu gleichzeitig aufgerufen, besteht die Möglichkeit, dass die Bedingung (anzahlGeräte > 0) für Thread 1 WAHR liefert; noch bevor Thread1 die Geräteanzahl dekrementiert, überprüft auch Thread2 die Bedingung, die dann ebenfalls WAHR ist. Erst anschließend verringern beide Threads die Variable um 1, was zu einer negativen Geräteanzahl führt. Eine der beiden Leihanfragen kann nicht bedient werden, obwohl die Buchung scheinbar erfolgreich war.

b Der Zugriff auf beide Methoden muss synchronisiert werden, etwa durch einen Monitor.

c Die Methode zum Abfragen der Entleihdaten, deren Ausführung die meiste Zeit benötigt, muss nicht innerhalb von *GerätAusleihen* (innerhalb des Monitors) ausgeführt werden. Sie könnte stattdessen nach erfolgreichem Abschluss der Methode *GerätAusleihen* aufgerufen werden.

d Es wurde aktives Warten umgesetzt: Die Methode *GerätAusleihen* wird in einer Wiederholung permanent solange aufgerufen, bis sie erfolgreich war. Ein solches Verfahren ist sehr ressourcenintensiv.

e Innerhalb der Methode *GerätAusleihen:*

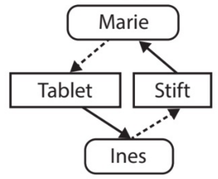
```
wiederhole solange anzahlGeräte <= 0
        Warten()
endewiederhole
anzahlGeräte = anzahlGeräte - 1
return WAHR
```

Innerhalb der Methode *GerätZurückgeben:*

```
anzahlGeräte = anzahlGeräte + 1
gerät.Freigeben()
Benachrichtigen()
```

f Alle Geräte bis auf einen Stift und ein Tablet sind bereits verliehen. Marie legt erst einen Stift und dann ein Tablet in ihren Warenkorb, Ines erst ein Tablet, dann den Stift. Beide Threads starten zeitgleich die Buchung. Für Marie wird der Stift gebucht, für Ines das Tablet. Beide Threads geben die gebuchte Ressource nicht mehr auf und warten auf die Ressource der jeweils anderen. Die Verklemmung wird nur dann aufgelöst, wenn andere Geräte zurückgegeben werden sollten.

Darstellung im Betriebsmittelzuteilungsgraphen:



g Die Verklemmung kann verhindert werden, indem man dafür sorgt, dass mindestens eine der vier CoffmanBedingungen nicht zutrifft: wechselseitiger Ausschluss, Halten und Warten, Ununterbrechbarkeit und zyklisches Warten.
Im Fall des Tabletverleihs könnte Halten und Warten verhindert werden, indem man das Programm so umstrukturiert, dass Warenkorbinhalte nicht nacheinander, sondern nur auf einmal gebucht werden. Der Gefahr einer zyklischen Wartesituation könnte begegnet werden, indem die Ressourcen in einer bestimmten Ordnung gebucht werden, z. B. erst die Tablets, dann die Stifte.

Stichwortverzeichnis

A
Abbruchbedingung 28, 89
Abschluss 32
abstrakte Klasse 10
abstrakte Methode 10
Advanced Persistent Threats 157
agile Methoden 160, 165
agile Werte 163
agiles Manifest 163
aktives Warten 129
Antivirensoftware 152
atomar 123
Attribut 7
Attributwert 7
Authentizität 149
AVL-Baum 85 f.

B
Backlog 161
Backtracking 97
– optimale Lösung 97
Backup 152, 157
Baum 52 ff.
– Höhe 53
Beobachtermuster 177
Betriebsmittelzuteilungsgraph 135
Bezeichner 7
Bibliothek 174
Binärbaum 54
– balanciert 55
– geordnet 54
– Implementierung 63 ff.
– vollständig 55
Blackbox-Prinzip 7
Blatt 52
Botnetz 157
Bundesamt für Sicherheit in der Informationstechnik (BSI) 150, 157
busy waiting 129

C
Capture the Flag 151
Coffman-Bedingungen 135
CPU 118
cyclic waiting 135

D
Dateizugriff 174
Datenbankverbindung 170
Datenbankzugriff 170
Datenbeschreibungsanweisungen 172
Datenmanipulationsanweisung 172
Datenschutz-Grundverordnung (DSGVO) 158
Datentyp 7
Deadlock 135
Debuggen 178
Deklaration 7
– leere Werte 8
Denial-of-Service (DoS) 157
design pattern 33
Distributed-Denial-of-Service (DDOS) 158
Drive-by-Exploits 158
Driver 162

E
Ebene 52
einfach verkettete Liste 26
entartet 55
Entwurfsmuster 33, 177
Entwurfsmuster Kompositum 33
Erzeuger-Verbraucher-Problem 128

F
Feld 9
Feldelement 9
FIFO-Prinzip 14
Firewall 152
Full Disclosure 154, 58

G
Generalisierung 9
geordneter Binärbaum 54
– Datenelement einfügen 60
– Datenelement suchen 60
– Grundstruktur 63
gezielte Angriffe 149
git 184
graphische Benutzeroberfläche 168

H
Hackback 155, 158
Halten und Warten 135
Hardwareausfall 153
Höhendifferenz 55
hold and wait 135

I
Index 9
inkrementell 160
Inorder-Durchlauf 71
Integrität 149
Interface 24
„ist ein“-Beziehung 9
Iteration 31, 160

K
Kante 52
Kernel 233
Klasse 7
KNOTEN 18
Knoten 52
– innerer 52
– Tiefe 53
Kompositum 33
Kompositum – Entwurfsmuster 177
Konstruktor 7
kritische Abschnitte 123

L
Lastenheft 164
leer 8
leere Werte 8
LIFO-Prinzip 45
lineare Rekursion 89
Liste 26
lokale Arbeitskopie 184

M
Malware 158
Man-in-the-Middle-Attacke 158
menschliche Fehlhandlungen 149
Merge-Konflikt 183
Methode 7
Methodenaufruf 7
Methodenkopf 7
Methodenrumpf 7
model 177
model-view-controller 177
Modularisierung 182
Monitor 124
MVC 175, 177

N
Navigator 162
Nebenläufigkeit 118
None (leere Werte) 8
non-preemption 135
null (leere Werte) 8

O
Oberflächenelemente 168
Oberklasse 9
Objekt 7
organisatorische Mängel 149
organisatorische Maßnahmen 153

P
Pair Programming 162
passives Warten 129
Passwortmanager 153, 158
Passwortregeln 153
Pentest 153, 158
Pfad 52
Pflichtenheft 164
Philosophenproblem 134
Phishing-E-Mails 150
physische Maßnahmen 152
Polymorphismus 9
Postorder-Durchlauf 72
Preorder-Durchlauf 71
Priorisierung 161
Programmfehler 178
– Bug 178
– Kompilierzeitfehler 178
– Laufzeitfehler 178
– logischer Fehler 178
Project Board 161
Prototyp 160
Prototyping 33
Prozess 119

R
Ransomware 154, 158
Rechenkern 118
Redundanz 182
Refactoring 182
Referenzattribut 8
Rekursion 89
– lineare 89
– verschachtelte 92
– verschränkte 92
– verzweigte 89
Rekursionsende 28, 89
Rekursionsschritt 28, 89
rekursiv 14
rekursive Beziehung 15
rekursive Datenstruktur 15
rekursive Methode 26
rekursiver Aufruf 97
Repository 183
Responsible Disclosure 154, 158
Retrospektive 161, 163
Review 163
Richtlinien 153

S
Sandbox 152, 158
Schnittstelle 8, 24
Schutzziele 149
Scraping 156, 158
Security Awareness 153, 155, 158
Security-Token 158
Sequenzdiagramm 8
Sicherheitsmaßnahmen
– organisatorische Maßnahmen 153
– physische Maßnahmen 152
– technische Maßnahmen 152
Social Engineering 153, 158
Spaghetticode 182
Spezialisierung 9
SQL-Anweisung 172
staging area 184
Stand-up-Meeting 162
Stapel 42
Synchronisation 122

T
Task 162
technische Maßnahmen 152
technisches Versagen 149
Test 162
– Black-Box-Test 180
– Integrationstest 180
– Modultest 180
– White-Box-Test 180
Testabdeckung 181
– Anweisungsabdeckung 181
– Branch-Abdeckung 181
– Funktionsabdeckung 181
– Zeilenabdeckung 181
Test-Driven-Development 181
testen 30, 164, 179
Thread 119
Tiefensuchbaum 105
Tiefensuche 103
– optimale Lösung 108 f.
traversieren 70
Trennung Struktur/Daten 18
Trennung von Struktur und Daten 18

U
überschreiben 9
Unterklasse 9
Ununterbrechbarkeit 135
User-Story 161

V
Verfügbarkeit 149
Verklemmung 135
verschachtelte Rekursion 92
Verschlüsselung 152
verschränkte Rekursion 92
Versionsverwaltungssysteme 183
Vertraulichkeit 149
verzweigte Rekursion 89
view 177
Vorgänger 55

W
Wartbarkeit 18, 182
Warteschlange 14, 42, 44
Wasserfallmodell 164
– mit Rückkopplung 165
wechselseitiger Ausschluss 123, 135
Wiederverwendbarkeit 18
Wurzel 52

Z
Zeitfenster 161
Zuverlässigkeit 155
Zwei-Faktor-Authentifizierung 155
zyklisches Warten 135

Bildquellenverzeichnis

Cover: Shutterstock.com/Olga Donchuk

Abbildungen/Fotos:
5/**11**/**12**/**40**/**67**/**70**/**73**/**78**/**104**/**110**/**111**/**127**/Shutterstock.com/Trueffelpix; **11**/mauritius images/alamy stock photo/BLM Photo; **18 o. re.**/Shutterstock.com/Focus and Blur; **50**/**52 Mi. re.**/**118 o. re.**/**185 u. re.**/Cornelsen/Inhouse; **93**/Shutterstock.com/Woodpond; **111 Mi. re.**/mauritius images/Memento; **115 o. li.**/stock.adobe.com/Christine; **115 o. re.**/mauritius images/Science Source; **115 u. li.**/sciencephotolibrary/Pasieka, Alfred; **115 u. re.**/Shutterstock.com/ KaryB; **122 o. re.**/Shutter-stock.com/qvist; **122 Mi. re.**/Shutterstock.com/ON-Photography Germany; **131 u. re.**/Shutterstock.com/FooTToo; **132 u. re.** /stock.adobe.com/Instantly; **154**/Shutterstock.com/Sasun Bughdaryan; **155**/Shutterstock.com/ra2 studio; **156**/stock.adobe.com/Berlin85; **179 u. re.**/Shutterstock.com/ptashka; **182 Mi.re.**/Shutterstock.com/MaraZe; **183 o. re.**/Graphics by N.D.Fernandez; **185 Mi. re.**/stock.adobe.com/Berthold Photography

Collagen:
22 o. re./Foto: Cornelsen/Inhouse; Zahn: Shutterstock.com/Maxx-Studio; **91 Mi.**/Cornelsen/Ingrid Schobel/Shutterstock.com/oixxo; **118**/**u. re.**/Rechenkerne: Shutterstock.com/asharkyu; Prozessor: Shutterstock.com/Vectorboost; **134 u. re.**/Cornelsen/inhouse/Shutterstock.com/Naci Yavuz/Ejichka/AAVAA; **51 o.**/**147 o.**/**147 Mi.**/Cornelsen/inhouse/Natascha Welz/Nicole Rademacher; **159 Mi.**/Foto: Cornelsen/Klaus Reinold; Ente: Shutterstock.com/ptashka; **160 Mi.**/Grafik: Cornelsen/Natascha Welz; Foto „Bug“: Shutterstock.com/trueffelpix.com; Foto Daumen: Shutterstock.com/ssd; Foto Cloud: Shutterstock.com/Blan-k; **177 Mi.**/Cornelsen/Ingrid Schobel/Shutterstock.com/dellaf semiga/Ma Sua; **178 Mi.**/Cornelsen/Nicole Rademacher/Dr. Stefan Seegerer; **183 u.**/Cornelsen/Ingrid Schobel/Nicole Rademacher; **186 Mi. re.**/Cornelsen/Reemers/Shutterstock.com/Marketa Kuchynkova/ stock.adobe.com/makrushka

Urheber der Bildschirmschüsse sind die Autoren.

Textquellenverzeichnis

S. 10/1	Bayerische Abiturprüfung 2022, Fach Informatik, Abschnitt II Aufgabe 1, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 11/2	Bayerische Abiturprüfung 2021, Fach Informatik, Abschnitt I Aufgabe 1, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 17/6	Bayerische Abiturprüfung 2013, Fach Informatik, Abschnitt II Aufgabe 2, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 23/5	Bayerische Abiturprüfung 2020, Fach Informatik, Abschnitt I Aufgabe 2a, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 45/5	17. Bundeswettbewerb Informatik 1999, Aufgabenblatt 1. Runde, © Bundesweite Informatikwettbewerbe (BWINF)
S. 46/T3	Bayerische Abiturprüfung 2022, Fach Informatik, Abschnitt II Aufgabe 2, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 47/T4	Bayerische Abiturprüfung 2021, Fach Informatik, Abschnitt I Aufgabe 2, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 57/9	Bayerische Abiturprüfung 2020, Fach Informatik, Abschnitt I Aufgabe 3, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 61/5	Bayerische Abiturprüfung 2018, Fach Informatik, Abschnitt I Aufgaben 1, 2, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/ schularten/gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 68/4	Bayerische Abiturprüfung 2019, Fach Informatik, Abschnitt I Aufgabe 2, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 69/6	Bayerische Abiturprüfung 2020, Fach Informatik, Abschnitt I Aufgabe 3, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 74/1	Bayerische Abiturprüfung 2021, Fach Informatik, Abschnitt I Aufgabe 3, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 74/2	Bayerische Abiturprüfung 2022, Fach Informatik, Abschnitt II Aufgabe 4, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 80/T3	Bayerische Abiturprüfung 2020, Fach Informatik, Abschnitt II Aufgabe 3, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 106/9	Bayerische Abiturprüfung 2016, Fach Informatik, Abschnitt I Aufgabe 5, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 107/10	Bayerische Abiturprüfung 2014, Fach Informatik, Abschnitt I Aufgabe 1, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 112/T3	Bayerische Abiturprüfung 2017, Fach Informatik, Abschnitt II Aufgabe 5, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/
S. 115	Benoît Mandelbrot. Die fraktale Geometrie der Natur; Birkhäuser Basel; April 2014; S. 13

Textquellenverzeichnis

S. 126/6 Bayerische Abiturprüfung 2017, Fach Informatik, Abschnitt III Aufgabe 4, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/

S. 138/5 Bayerische Abiturprüfung 2016, Fach Informatik, Abschnitt III Aufgabe 2, Quelle: Staatsinstitut für Schulqualität und Bildungsforschung (ISB), https://www.isb.bayern.de/schularten/ gymnasium/leistungserhebungen/abiturpruefung/informatik/

INFORMATIK 6

Listen | Bäume | Rekursion
Nebenläufigkeit und Threads
Informationssicherheit
Softwareprojekte

Cornelsen

ISBN 978-3-637-02476-2